Control de temperatura y humedad de una maqueta de invernadero mediante tecnología Open Source

Juan José Puerta Muñoz

8 de Octubre de 2014

Índice general

Me gustaría agradecer enormemente a Julio Ibarrola y José Manuel Cano por mostrarse casi tan implicados como yo en la realización de este proyecto. Y por ayudarme cada vez que podían y aportar nuevas ideas. A Pablo Molina por su ayuda a pesar de los errores de novato que cometía una y otra vez en el laboratorio. A Carlos por sacarme alguna vez de algún apuro. A toda mi familia por apoyarme en estos momentos y confiar en mí. Y por último a mi pareja por tener la paciencia de aguantar cada día de trabajo que he pasado realizando este proyecto y me ha quitado tiempo de estar con ella.

Muchas gracias a todos...

Capítulo 1

Introducción y objetivos.

1.1. Antecedentes

Unos antiguos alumnos del Dpto. de ingeniería de sistemas y automática realizaron una maqueta con el objetivo de emular el comportamiento de un invernadero. Además del diseño del dispositivo, realizaron una tarjeta de adquisición de datos para el manejo y el control de los actuadores de este. El resultado final fue el control de temperatura mediante Matlab y un PIC 18F255.

El nuevo trabajo a realizar inicialmente consistía en cambiar la tarjeta de adquisición por una con Arduino Uno, automatizar la trampilla y conseguir un control multivariable de temperatura y humedad. Debido también al auge creciente de la tecnología Open Source, se estudió la posibilidad de realizar todo el trabajo con este tipo de tecnología un poco ausente durante mi trayectoria académica. A partir de ahí se comenzó a investigar las distintas alternativas que podíamos utilizar en nuestro proyecto. Para completar y hacer más operativa la maqueta se ha pensado en automatizarla mediante un Arduino Uno y monitorizarla a través de Scilab.

1.2. Objetivos del proyecto.

El objetivo es completar y automatizar la maqueta de invernadero mediante software Open Source. Para este objetivo necesitamos realizar las siguientes tareas:

- **Explicación y mención de un abanico de alternativas Open Source en el ámbito de la ingeniería:**
  Daremos una definición de en qué consiste la tecnología Open Source y haremos un recorrido en los proyectos más interesantes. Estudiaremos también las principales alternativas Open Source al software que necesitamos para nuestro proyecto, haciendo especial hincapié en Matlab.

- **Explicación detallada de los componentes de nuestra maqueta:**
  En este capítulo daremos una descripción detallada del hardware que hemos utilizado: actuadores,

sensores, etc. Veremos lo que llevaba la maqueta en el proyecto anterior, desarrollaremos con detalle los cambios realizados y examinaremos los nuevos elementos añadidos.

- **Diseño de la tarjeta de control necesaria para la adquisición y control de nuestra maqueta:**
  En este capítulo mostraremos la electrónica necesaria para el diseño de nuestra tarjeta de control y adquisición. Mostraremos también, paso a paso, el proceso para la obtención de la PCB fotosensible.

- **Desarrollo del software necesario para nuestro proyecto:**
  En este apartado realizaremos una explicación del todo el software desarrollado para conseguir el pleno funcionamiento de nuestra maqueta. Describiremos las funciones y librerías de Arduino necesarias, las funciones de Scilab y el entorno Xcos donde programaremos el control de nuestra maqueta.

- **Experimentación y Control:**
  En este capítulo se mostraran los resultados obtenidos de las identificaciones y los distintos sistemas de control desarrollados. Se realizará el control de la temperatura y humedad por separado, el control multivariable y, por último, el control multivariable desacoplado. Adjuntaremos las gráficas y se comentaran los resultados obtenidos.

Capítulo 2

Open Source en la ingeniería.

2.1. Introducción

En un mundo globalizado, interconectado constantemente, la tecnología Open Source ha cambiado la forma de entender la ingeniería. Donde la inteligencia colectiva coge forma y cada uno pone su granito de arena en ese monstruo del conocimiento llamado Internet.

El abaratamiento del coste de plataformas hardware y el acceso a Internet han dado al ciudadano medio la capacidad de poder realizar proyectos que antes estaban al alcance de unos pocos. Poco dinero, Internet y ganas es la receta para que cualquier persona sea capaz de realizar proyectos y trabajos que en épocas pasadas eran una quimera. Gracias a estas plataformas, ingenieros de todo el mundo podrán realizar controles PID, microrobots, domótica y un sinfín de aplicaciones a muy bajo coste con un buen rendimiento y además subirla a la red para que alguien pueda consultar, modificar e incluso mejorar ese proyecto.

Pero ¿Qué significa Open Source? Open Source es un término utilizado para definir un software al que se tiene pleno acceso de su código fuente, pero según la Open Source Iniciative (Figura 2.1) no se limita solo a eso, sino que además debe cumplir una serie de requisitos[1] .

[1] Decálogo traducido de: `http://opensource.org/osd`

Figura 2.1: Logo OSI

1. **Distribución libre:**
   La licencia no debe restringir a un tercero el vender o entregar el programa como parte de una distribución mayor que contiene programas de diferentes fuentes. La licencia no debe requerir un royalty u otras comisiones para esta venta.

2. **Código fuente:**
   El programa debe incluir el código fuente, y debe permitir la distribución en código fuente y en forma compilada. Si un producto no se distribuye con el código fuente, debe de haber una manera sencilla de acceder a él y de distribuirlo,como por ejemplo, la descarga a través de Internet sin cargo. El código fuente debe ser la forma preferida en la cual un programador podría modificar el programa.

3. **Trabajos derivados:**
   La licencia debe permitir modificaciones y trabajos derivados, y debe permitir que estos se distribuyan bajo los mismos términos que la licencia del software original.

4. **Integridad del código fuente del autor:**
   La licencia puede restringir que el código fuente se distribuye en forma modificada sólo si la licencia permite la distribución de "archivos parche" con el código fuente con el fin de modificar el programa en tiempo de construcción. La licencia debe permitir explícitamente la distribución de software a partir de código fuente modificado. La licencia puede requerir trabajos derivados a llevar un nombre o número de versión diferente del software original.

5. **No discriminación contra personas o grupos.**

6. **No discriminación en función de la finalidad perseguida:**
   La licencia no debe restringir a nadie que haga uso del programa en un campo específico de actividad.

7. **Distribución de la licencia:**
   Los derechos asociados al programa deben aplicarse a todos aquellos a quienes se redistribuya el programa, sin necesidad de pedir una licencia adicional para estas terceras partes.

8. **La licencia no debe ser específica para un producto:**
   Los derechos asociados al programa no deben depender de que parte del programa de distribución de software en particular. Si el programa se extrae de esa distribución y usado o distribuido dentro de los términos de la licencia del programa, todas las partes a las que se redistribuye el programa deben tener los mismos derechos que los que se conceden con la distribución de software original.

9. **La licencia no debe Restringir Otro Software:**
   La licencia no debe poner restricciones sobre otros programas que se distribuyan junto con el software licenciado. Por ejemplo, la licencia no puede insistir que todos los demás programas distribuidos sobre el mismo medio deben ser software de código abierto.

10. **La licencia debe ser tecnológicamente neutral:**
    Ninguna disposición de la licencia puede basarse en cualquier tecnología o estilo de interfaz individual.[2]

2.2. Aportaciones del Open Source.

Por momentos puede parecer que al hablar del software Open Source estemos hablando de un fenómeno minoritario sin mayor importancia en la industria, pero este tipo de software está más que extendido en nuestra vida cotidiana. Aquí realizaremos una breve descripción de los aportes más importantes del Open Source.

1. **Ubuntu (Figura 2.2):**[3]
   El nombre proviene del concepto africano Ubuntu, que significa "humanidad hacia otros" o "yo soy porque nosotros somos". También es el nombre de un movimiento humanista sudafricano. Esta distribución puede que sea la más famosa en el mundo del PC. Ubuntu ha dado a Linux el salto de calidad necesario para que la experiencia de usar una distribución de este tipo sea para todo tipo de público. Además, ha conseguido acuerdos con empresas de hardware como HP o Nvidia para el diseño de drivers propios, mejorando así su rendimiento. También ha entrado en la carrera por los smartphones desarrollando Ubuntu Touch. El coste de descarga de estas distribuciones es gratuito y ofrecen un soporte de 5 años mediante actualizaciones.

[2] Decálogo traducido de: `http://opensource.org/osd`
[3] Información obtenida de `http://www.ubuntu-es.org/`

Figura 2.2: Logo Ubuntu

2. **Debian (Figura 2.3):**[4]
   Debian o Proyecto Debian es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU para PC basado en software libre. Con una estructura jerarquizada, los desarrolladores de Debian tienen que aceptar las reglas establecidas por "El Contrato Social de Debian", "Las directrices de Software Libre" y "La Constitución de Debian". En estas reglas aparece la opción de elegir al líder del proyecto una vez al año, que es el que ordena las fechas de lanzamiento. Debian no proporciona software privativo de forma explícita como Ubuntu, pero proporciona repositorios extra por si el usuario quiere o necesita para el desarrollo de una tarea. Ubuntu está basado en Debian. Debian no vende su software pero si permite que empresas lo hagan y saquen un beneficio.

Figura 2.3: Logo Debian

3. **Android (Figura 2.4):**[5]
   Sistema Operativo basado en el kernel de Linux desarrollado por Google y orientado a smartphones y tablets. Gracias a su sencillo desarrollo y el bajo coste que supone el diseño de apps para

[4]Información obtenida de: `https://www.debian.org/`
[5]Información obtenida de `http://es.wikipedia.org/wiki/Android`

esta plataforma han hecho de Android el sistema con mejor entorno de aplicaciones y el favorito de los usuarios, copando casi el 85 % de cuota de mercado.[6]

Figura 2.4: Logo Android

4. **VideoLAN (Figura 2.5):** [7]
   VideoLAN es una solución de software completa para transmisión de vídeo, desarrollada por estudiantes de École centrale Paris y desarrolladores de todo el mundo, dentro de GNU General Public License (GPL). VideoLAN está diseñado para transmitir vídeo MPEG en redes con gran capacidad de ancho de banda. VLC es un reproductor multimedia libre y de código abierto multiplataforma y un framework que reproduce la mayoría de archivos multimedia, así como DVD, Audio CD, VCD y diversos protocolos de transmisión.

Figura 2.5: Logo VLC

[6]"Android copa el 85 % de cuota de mercado de smartphones"`http://www.europapress.es/portaltic/sector/noticia-android-copa-85-cuota-mercado-smartphones-20140731174435.html`

[7]Información obtenida de `http://www.videolan.org/index.es.html`

5. **Mozilla Firefox y Firefox OS (Figura 2.6):**[8]
   La Fundación Mozilla es una organización sin ánimo de lucro dedicada a la creación de software libre y a la innovación en la red; y sus dos productos estrella son Mozilla Firefox y Firefox OS. Mozilla Firefox es un navegador web libre y de código abierto desarrollado para Microsoft Windows, Mac OS X y GNU/Linux coordinado por la Corporación Mozilla y la Fundación Mozilla. Firefox OS es un S.O. lanzado para smartphones basado en HTML5 pensado para smartphones de bajo coste en mercados emergentes.

Figura 2.6: Logo Firefox

6. **Blender (Figura 2.7)**: [9]
   Blender es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. También de composición digital utilizando la técnica procesal de nodos, edición de vídeo, escultura (incluye topología dinámica) y pintura digital. En Blender, ademas, se puede desarrollar vídeo-juegos ya que posee un motor de juegos interno.

Figura 2.7: Logo Blender

[8] Información obtenida de `https://www.mozilla.org/es-ES/about/`
[9] Información traducida de `http://www.blender.org/about/`

7. **Scilab**: [10]
   Scilab (Figura 2.8) es un software matemático, con un lenguaje de programación de alto nivel, para cálculo científico, interactivo de libre uso y disponible en múltiples sistemas operativos (Mac OS X, GNU/Linux, Windows). Es el sustituto natural de Matlab en Open Source, ya que dispone una herramienta de análisis de sistemas dinámicos llamada Xcos, muy similar a Simulink. Scilab será la parte más importante de nuestro proyecto y hablaremos sobre él más adelante.

Figura 2.8: Logo Scilab

2.3. Hardware Open Source u Open Hardware

Compartiendo la filosofía del software Open Source, nos encontramos el Open Hardware. El "Hardware Open Source" es un término para denominar artefactos tangibles: máquinas, dispositivos, u otros objetos del mundo físico; cuyo diseño ha sido publicado de forma tal que cualquier persona pueda fabricar, modificar, distribuir y usar esos objetos. Esta definición tiene la intención de proveer una guía para el desarrollo y evaluación de licencias para "Hardware de fuentes abiertas". El Open Hardware es relativamente más moderno que el software Open Source, por lo que los proyectos todavía no han tenido un desarrollo muy avanzado. En la mayor parte de los proyectos, la financiación se realiza mediante Crowfunding o donaciones. Aun así encontramos proyectos interesantes que pueden tener un largo recorrido en el mercado.

1. **Arduino (Figura 2.9)**: [11]
   Proyecto estrella del Open Hardware, Arduino es un entorno de desarrollo basado en una placa con un microcontrolador, diseñada para facilitar el uso de la electrónica. Arduino puede tomar información del entorno a través de sus entradas analógicas y digitales, puede controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante

[10] Información extraída de `http://es.wikipedia.org/wiki/Scilab`
[11] Información traducida de `http://www.arduino.cc/`

el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Existen diferentes modelos de placas Arduino, cada una adecuada para un presupuesto o un proyecto específico. Los scripts hechos con Arduino pueden ejecutarse sin necesidad de conectar a un computador.

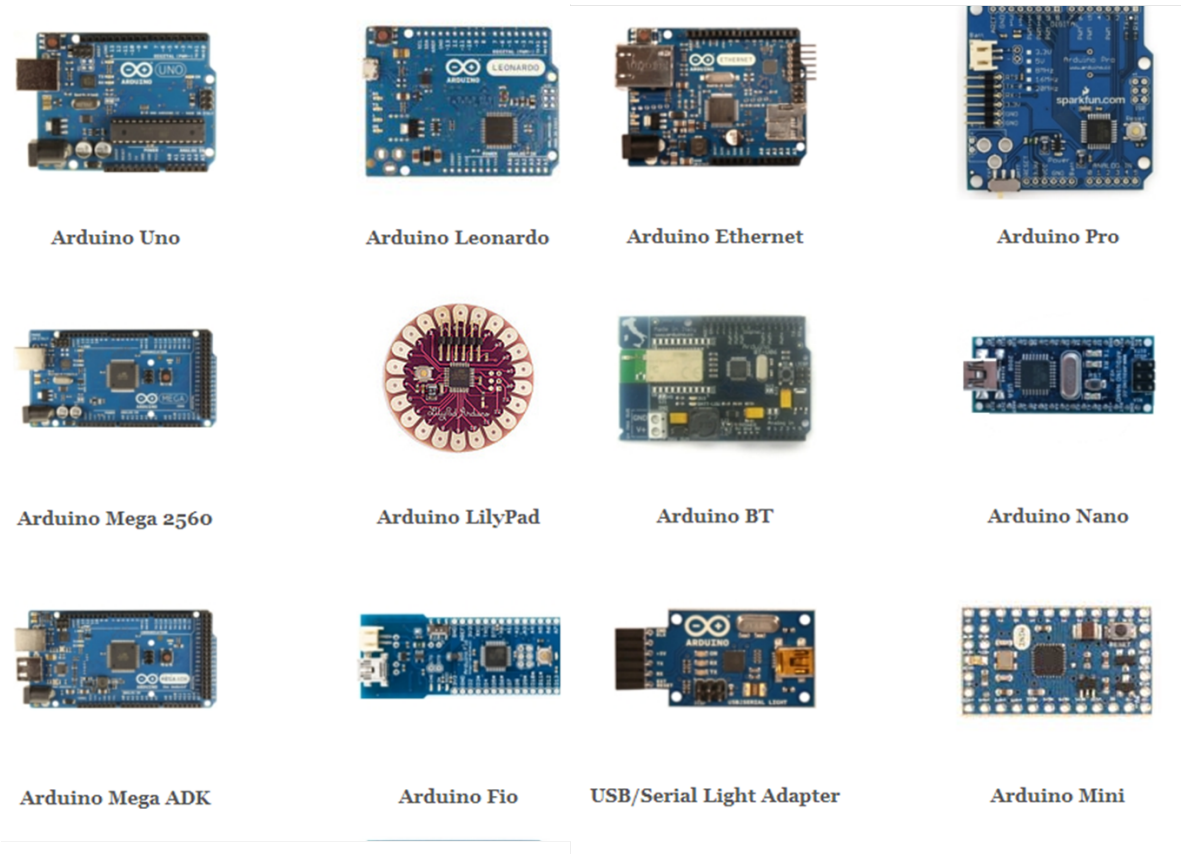


Figura 2.9: Diferentes modelos Arduino

2. **Proyecto RedRap**: [12]
   El Proyecto Reprap (Figura 2.10) es una iniciativa con el ánimo de crear una máquina autoreplicable que puede ser usada para prototipado rápido y manufactura. Una máquina de prototipado rápido es una impresora 3D que es capaz de fabricar objetos en tres dimensiones a base de un modelo hecho en ordenador.

[12] Información obtenida de `http://reprap.org/wiki/RepRap/es`

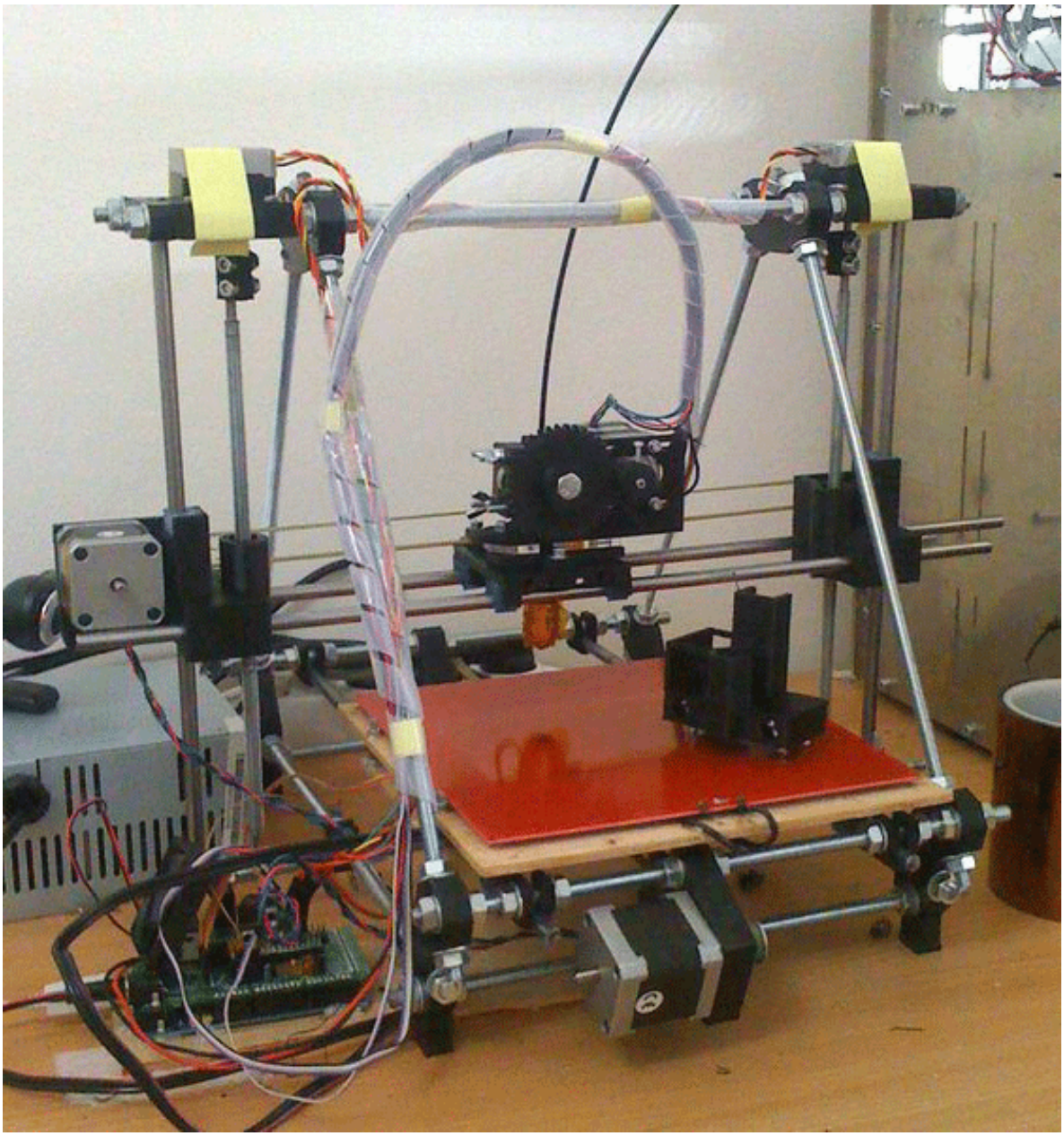


Figura 2.10: Redrap

3. **Red Pitaya (Figura 2.11)**: [13]
   Se trata de un hardware con todo lo necesario para que uses un PC, una tablet o un smartphone como equipo de investigación común de un laboratorio. Con algunos cables para conectarlo y unos clicks podrás comenzar a usar un osciloscopio, un analizador de espectro y un analizador

[13] Puntos 3,4,5 extraídos de este artículo `http://gizmologia.com/2014/01/hardware-open-source`

de frecuencia de respuesta.

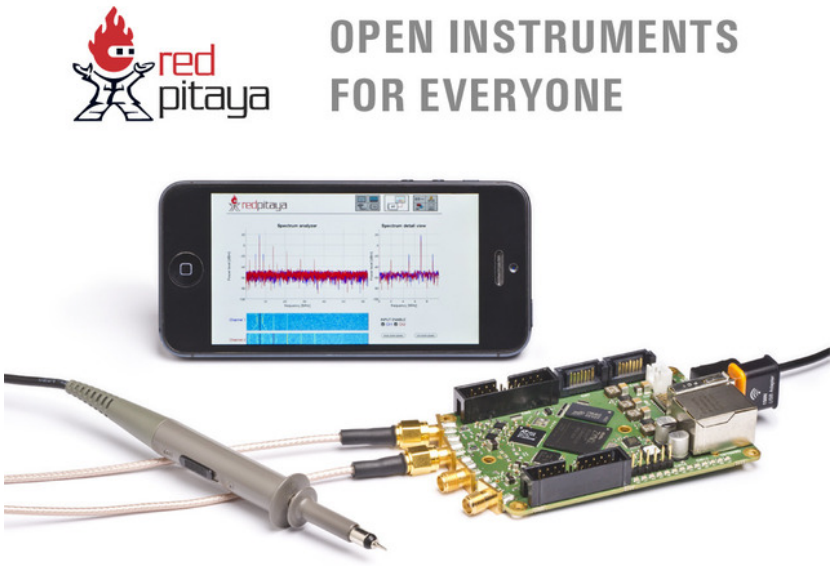


Figura 2.11: Red Pitaya

4. **Poppy**:
   El proyecto Poppy (Figura 2.12) tiene como objetivo la construcción de un humanoide Open Source. Diseñado por el Flowers Lab en el INRIA Burdeos y Ensta ParisTech (Francia), su desarrollo tiene como objetivo proporcionar un robot humanoide asequible y modificable para la ciencia, la educación, el arte y los amantes de la robótica. El robot cuesta unos 7500€ y viene con todas las partes diseñadas en una impresora 3D, junto a los motores y la electrónica.

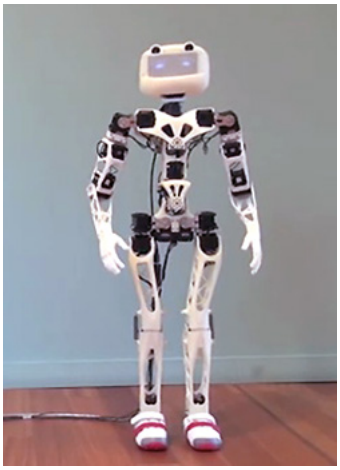


Figura 2.12: Fotograma Poppy

5. **Phoneblocks**:
   Phoneblocks (Fig 2.13) es un proyecto que consiste en el diseño de un smartphone modular, en el que cada usuario podrá elegir el smartphone a su medida añadiendo y quitando módulos. Mejorando así el mantenimiento del teléfono y combatir la obsolescencia programada. Empresas como Motorola están ahora muy interesadas en el proyecto y van a dar lugar al Motorola Ara.[14]

Figura 2.13: Phonebloks

6. **OScar**: [15]
   Este proyecto es el intento de diseñar un coche mediante las reglas del Open Source. Cualquier ingeniero, diseñador o especialista puede participar en el diseño del automovil u otros diseños planeados por la empresa. Debido a las limitaciones, OScar (Figura 2.14) es de momento modelado por ordenador a partir de software CAD de código abierto, a la espera de elaborar a continuación prototipos tridimensionales. La plataforma OScar todavía no ha producido un automóvil, aunque ya existe abundante documentación y especificaciones.

[14]"Google se queda con el Motorola Project Ara de teléfonos modulares"`http://www.elandroidelibre.com/2014/01/google-se-queda-con-el-motorola-project-ara-de-telefonos-modulares.html`

[15]Información obtenida de `http://www.theoscarproject.org/`

Figura 2.14: Imagen OScar

2.4. La comunidad en el software propietario.

El modelo del Open Source, caracterizado principalmente por el trabajo comunitario, ha hecho que el desarrollo de estas plataformas haya sido muy rápido y eficaz. Solo hay que ver como ha sido el crecimiento de Android, Ubuntu o Arduino en los últimos años. Esto ha hecho que compañías de software propietario como Autodesk, National Instruments o Mathwork han creado páginas web (Figura 2.15) y foros en los que los desarrolladores pueden crear mejoras, compartir o vender distintas aplicaciones. Este tipo de estrategia agiliza el desarrollo y la mejora del software ahorrando costes a la empresa.

Debido también al crecimiento del Software Open Source y del Open Hardware, las empresas de software propietario han ofrecido interacción entre software debido a la demanda de los usuarios. Así por ejemplo podemos encontrar casos como estos:

- Soporte para Arduino en Matlab y Simulink[16]: Matlab proporciona gratis una serie de sketch de Arduino para manejarlo desde Matlab o Simulink. Así podremos ejecutar funciones de lectura y escritura programando directamente desde la interfaz de Matlab.
- Scilab Gateway:[17] Esta extensión de Labview permite invocar todos los comandos de Scilab en nuestros VIs de Labview.

2.5. Buscando una alternativa a MATLAB.

Todo aquel que está estudiando o está trabajando en el mundo de la ingeniería sabe de la importancia de Matlab. Matlab puede ser el software matemático más extendido en el ámbito universitario, en el de la industria y en el de la investigación. Pero ¿Qué es Matlab y por qué está tan extendido?

MATLAB[18] (abreviatura de MATrix LABoratory, "laboratorio de matrices") es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de

[16] Información obtenida de `http://www.mathworks.es/hardware-support/arduino-matlab.html`
[17] Información obtenida de `https://www.scilab.org/scilab/interoperability/connection_applications/labview`
[18] Información extraída de `http://es.wikipedia.org/wiki/MATLAB`

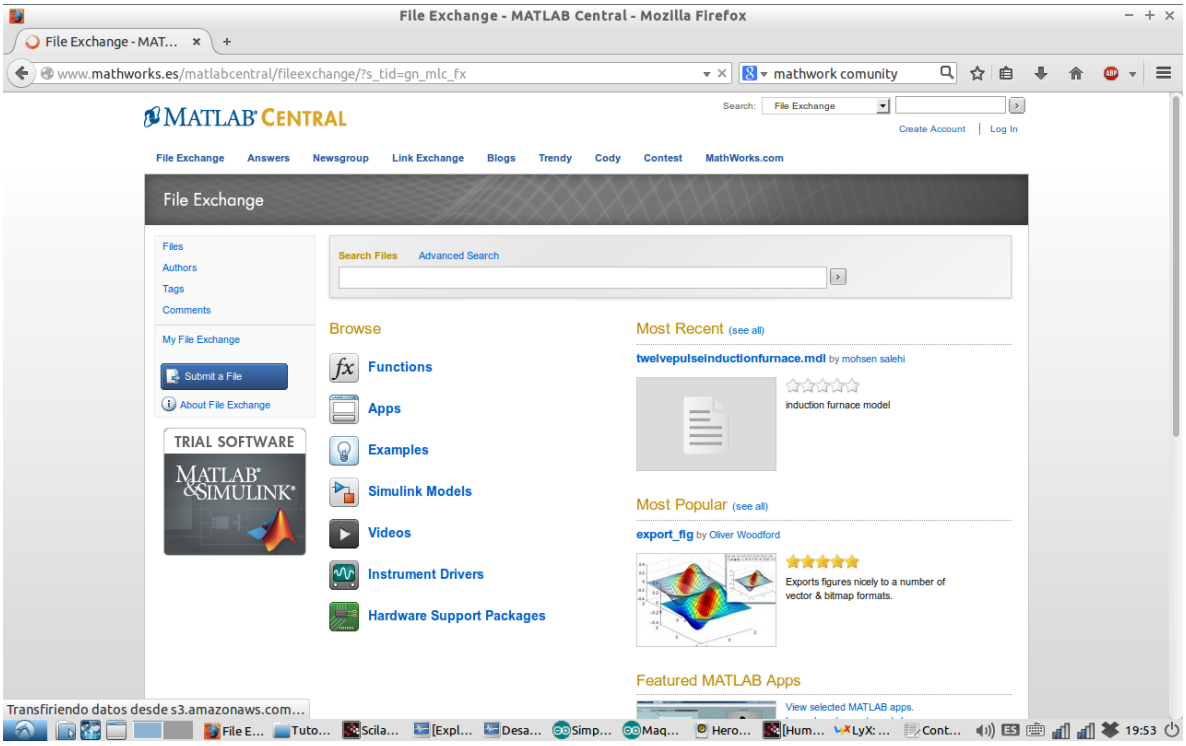


Figura 2.15: Web de la comunidad Matlab

programación propio (lenguaje M) y servicio de especie. Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux .

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete Matlab dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de Matlab con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets). Es un software muy usado en universidades y centros de investigación y desarrollo. En los últimos años ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal o crear código VHDL.

Debido a la capacidad multidisciplinar de Matlab y su potencia, es difícil encontrar un software Open Source con una potencia al menos similar y con un desarrollo también similar al software de Mathwork.

En nuestro caso hemos encontrado dos alternativas aceptables a Matlab: GNU Octave y Scilab.

2.5.1. GNU Octave:

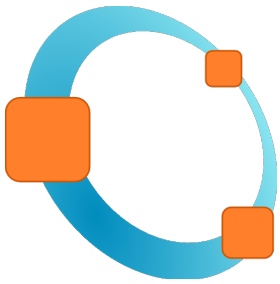


Figura 2.16: Logo GNU Octave

2.5.1.1. ¿Qué es GNU Octave?

Es el equivalente a Matlab en cuanto a programación. Puede realizar scripts en formato .m y su compatibilidad con Matlab es casi perfecta. GNU Octave (Fig 2.16 y Fig 2.17) es un software perteneciente al Proyecto GNU, fundado por el informático Richard Stallman en 1983. El objetivo de GNU consiste en el la realización de un S.O totalmente libre sin ningún tipo de software propietario.

GNU Octave es un lenguaje de alto nivel interpretado, destinado principalmente para cálculos numéricos. Proporciona capacidades para la solución numérica de problemas lineales y no lineales, y para realizar otros experimentos numéricos. También proporciona amplias capacidades de gráficos para la visualización de datos y la manipulación.[19]

GNU Octave ofrece también una serie de paquetes o extensiones para aplicaciones específicas como lecturas de puerto serie o gráficos especiales.

[19]Información extraída de `http://www.gnu.org/software/octave/`

2.5.1.2. Virtudes y Defectos de GNU Octave.

Entre las virtudes de GNU Octave podemos destacar:[20]

- Gran compatibilidad con Matlab: Los scripts de GNU Octave se escriben en .m, como en Matlab. La compatibilidad entre ambos sistemas es muy alta. Puedes sejecutar con ambos programas sin cambiar una coma:

1. Programación imperativa con los comandos if-else, for, while;
2. Importar y exportar datos; estructuras de datos y matrices;
3. Tratamiento de imágenes.

- Precio: GNU Octave es gratuito y es compatible con Windows, Linux y MAC OS.
- Memoria: GNU Octave consume muy pocos recursos y ocupa muy poca memoria en el disco duro.

Entre los principales defectos que podemos encontrar en GNU Octave destacamos:

- La ausencia de GUI propia: La GUI de Octave está en desarrollo, de momento solo está en fase beta en la versión 3.8.1. en distribuciones Linux, en Windows no está disponible. Se espera que en la versión 4.0 esté lista.
- Poco desarrollo: El desarrollo de Octave me ha parecido algo pobre para los objetivos de nuestro proyecto. No hay aplicaciones para crear GUI, y además carece de algo parecido a Simulink, lo que ahorraría bastante trabajo y haría nuestro proyecto mejor visualmente.

2.5.1.3. Conclusión

GNU Octave es una herramienta potente y una buena alternativa si lo que necesitas es cálculo matemático y el dibujo de gráficas. Sin embargo, hemos descartado el uso de GNU Octave para nuestro proyecto por su pobre desarrollo en temas relacionados con la ingeniería de control y con la interacción con sistemas físicos.

2.5.2. Scilab

2.5.2.1. ¿Qué es Scilab?

Ya hemos hablado de Scilab (Figura 2.18) en el capítulo anterior pero ahora vamos a extendernos algo más.

Scilab es un software matemático, con un lenguaje de programación de alto nivel, para cálculo científico, e interactivo de libre uso desarrollado por el INRIA en 1990. Actualmente desarrollado por

[20] Información extraída de `https://www.slideshare.net/herraiz/matlab-yo-uso-octave-upm`

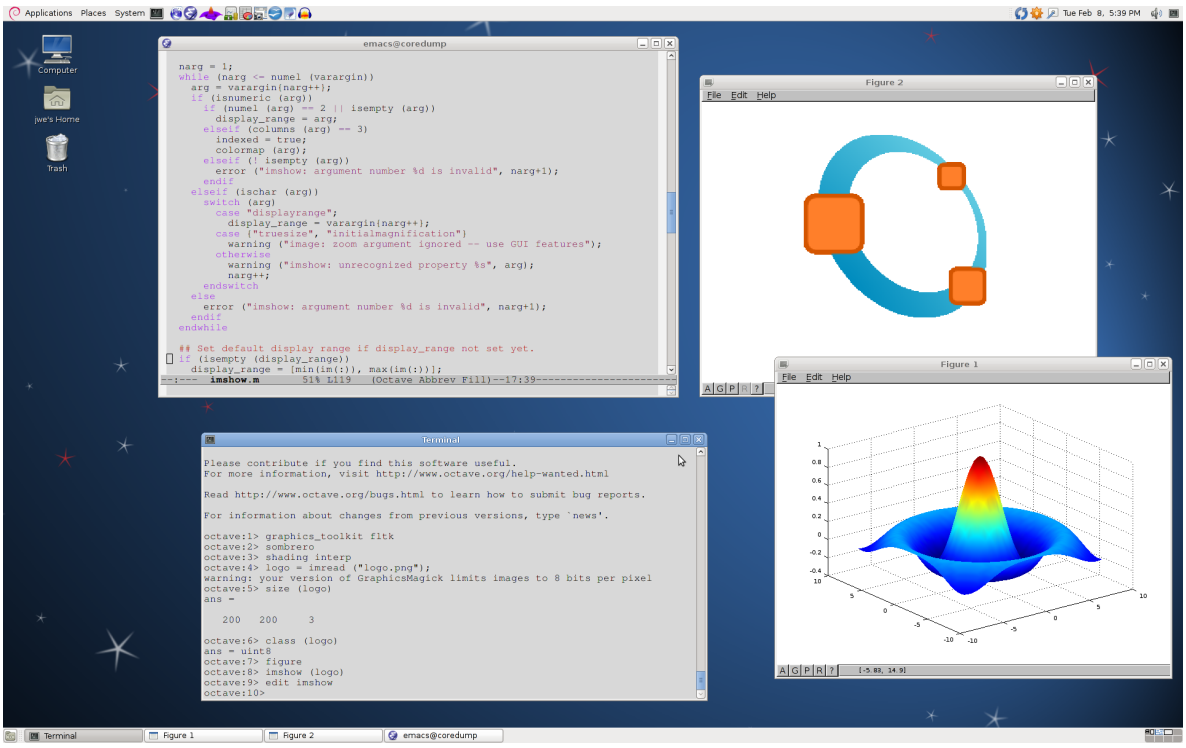


Figura 2.17: GNU Octave en Debian

Scilab Enterprise desde 2012. Scilab fue creado para hacer cálculos numéricos, aunque también ofrece la posibilidad de hacer algunos cálculos simbólicos como derivadas de funciones polinómicas y racionales. Posee cientos de funciones matemáticas y la posibilidad de integrar programas en los lenguajes más usados (Fortran, Java, C y C++).

Scilab viene con numerosas herramientas: gráficos 2-D y 3-D, animación, álgebra lineal, matrices dispersas, polinomios y funciones racionales, simulación, programas de resolución de sistemas de ecuaciones diferenciales (explícitas e implícitas) y una de las herramientas más interesantes, Xcos.

Xcos un simulador de sistemas por álgebra de bloques muy parecido a Simulink. Con Xcos podremos realizar sistemas dinámicos, control clásico, robusto, etc.

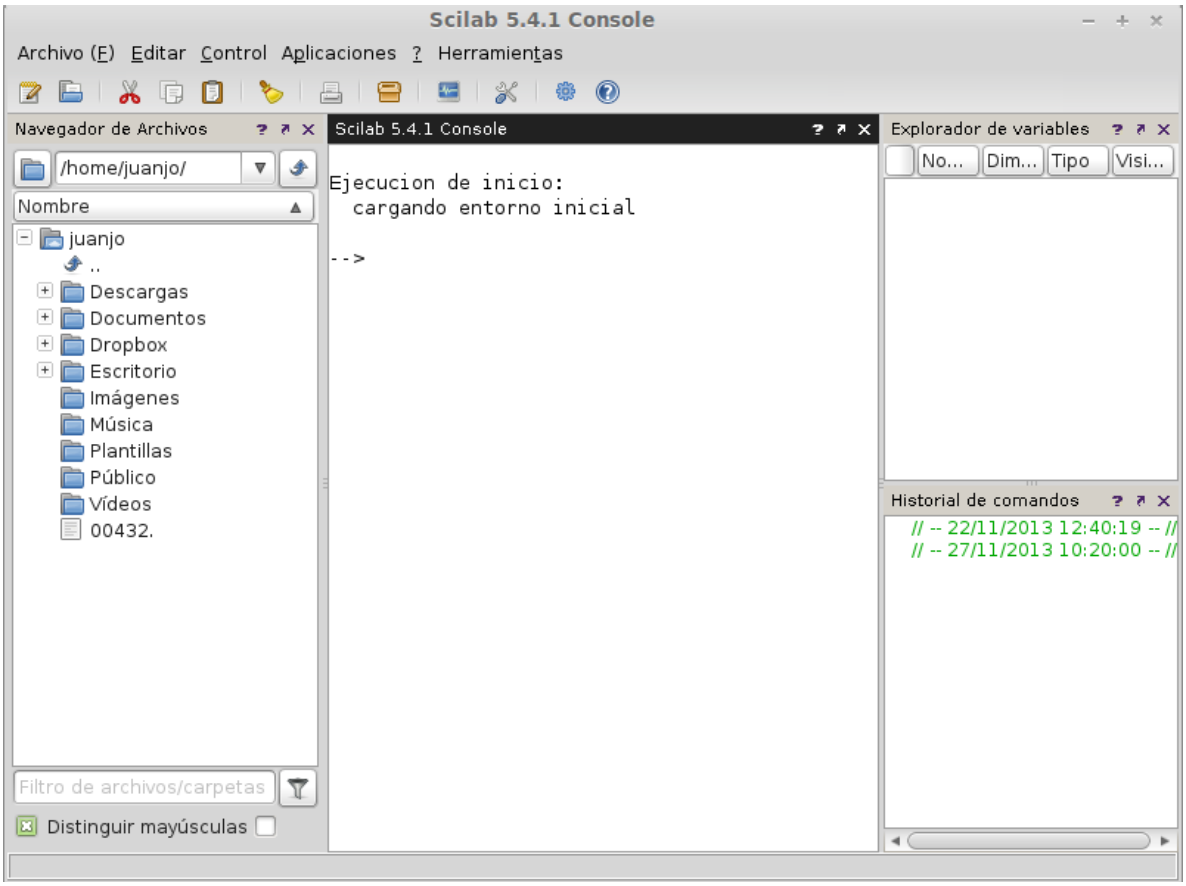


Figura 2.18: Interfaz Scilab

2.5.2.2. Virtudes y defectos de Scilab.

Entre los puntos favorables de Scilab podemos destacar:

- Mejor desarrollo que Octave: Mediante los ATOMS (Figura 2.19) o módulos, Scilab tiene herramientas bastante útiles que hacen que la programación sea más sencilla. Entre los módulos

más destacados encontramos una aplicación para diseñar GUI de forma gráfica, funciones para comunicación serie, bloques en Xcos para manejar Arduino, identificación de sistemas, etc.

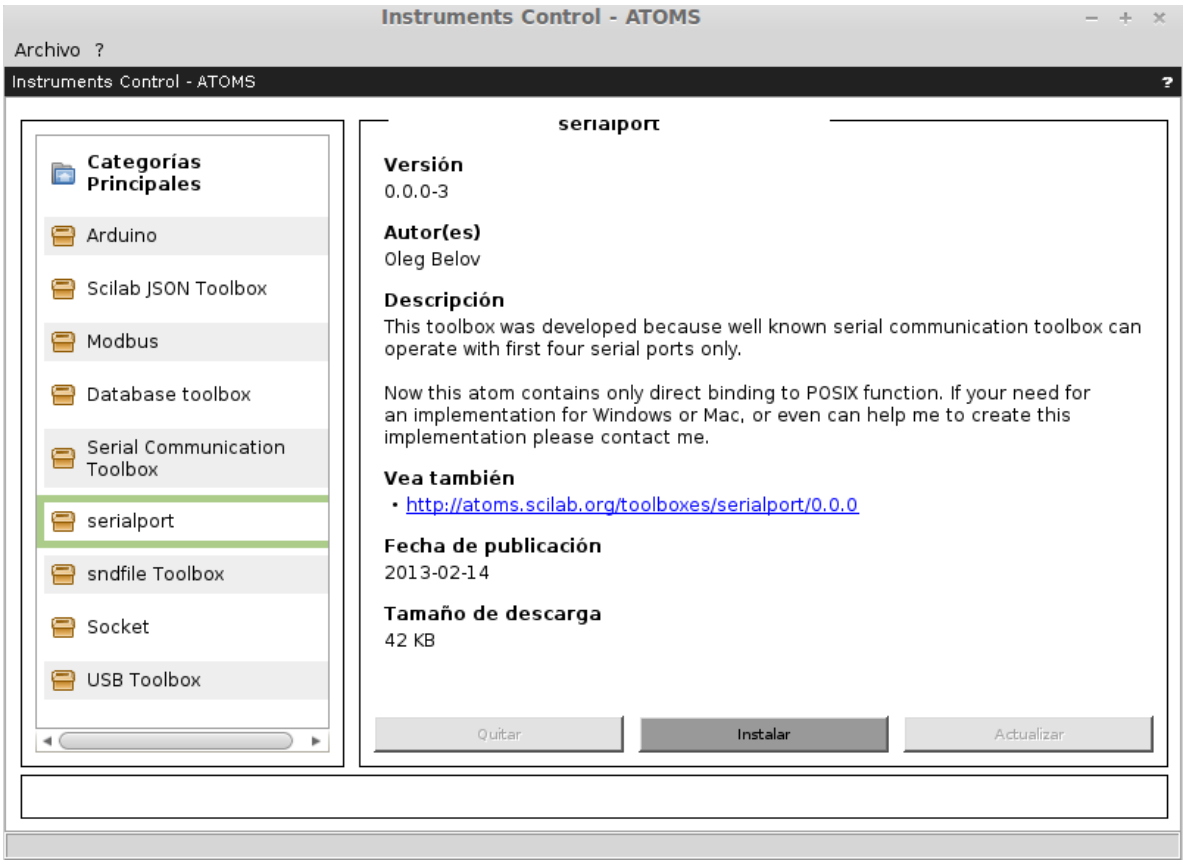


Figura 2.19: Interfaz instalación ATOMS

- Xcos (Figura 2.20): Sin duda la herramienta que ha hecho que nos decantemos por Scilab para el desarrollo de nuestro proyecto. Gracias a Xcos la programación de nuestro sistema de control es más sencilla, rápida e intuitiva.

Entre los puntos en contra de Scilab podríamos encontrar:

- Lenguaje distinto de Matlab: Aunque siendo parecida la sintaxis de Scilab a la de Matlab, no hay compatibilidad entre ambos. Scilab maneja un lenguaje distinto, con lo que tendremos que aprender el lenguaje de Scilab para hacer lo que hacíamos en Matlab. En la sección de ayuda de Scilab existe una apartado donde se pueden ver equivalentes de funciones de Matlab en Scilab.
- Desarrollo todavía insuficiente: Aunque mejora a GNU Octave, Scilab está todavía a años luz del desarrollo de Matlab actualmente. Xcos todavía no tiene funciones desarrolladas que en Simulink ya están implementadas y módulos como el de identificación de sistemas son muy pobres comparados con los de Matlab.

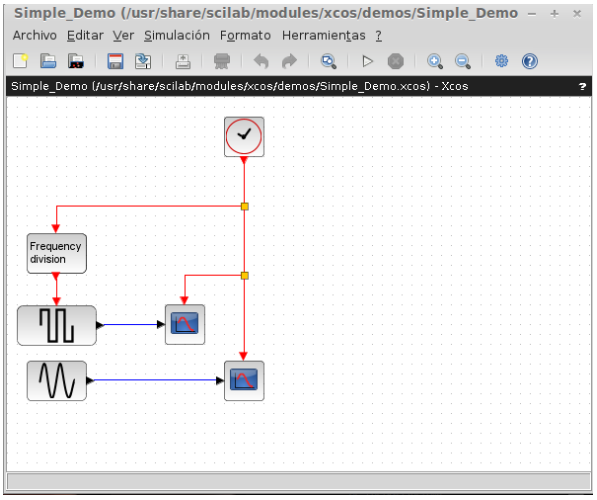


Figura 2.20: Interfaz Xcos

2.5.2.3. Conclusión.

Scilab se encuentra, en cuanto a desarrollo, en el término medio entre GNU Octave y Matlab. Para aplicaciones sencillas es muy útil y un gran sustituto para la docencia. La comunicación con sistemas físicos también es buena. Sin embargo, la cantidad de desarrollo que tiene Matlab a sus espaldas hace que, en términos de productividad, Matlab sea mejor.

A pesar de eso, Scilab será el software elegido para el desarrollo de nuestro proyecto, ya que dispone de las herramientas suficientes para realizarlo. Demostrando así la capacidad del Open Source para realizar proyectos de control y automatización.

2.6. Open Source utilizado en nuestro proyecto.

Además de Scilab y Arduino, el resto de software que vamos a utilizar en este proyecto será Open Source. Se ha buscado por la red cantidad de alternativas a editores de texto y CAD de electrónica que se adecuen a nuestras necesidades. Hemos seleccionado las dos que mejor se adaptaban a nuestro proyecto.

1. **LYX:**
   LYX (Figura 2.21) es un editor de texto que tiene la capacidad de compilar en TEX/LATEX disponible para Windows, GNU/Linux y Mac Os distribuido con licencia Open Source. Combina la potencia de este código con una interfaz gráfica en la que podemos escribir directamente nuestro texto. Permite la integración de bibliografias con JabRef, lenguaje matemático y científico. Además de dar su salida en PDF con una calidad dificilmente igualable en otros editores de texto. Tiene también plantillas de documentos de universidades o asociaciones, como la American Chemical Society y la American Mathematical Society.

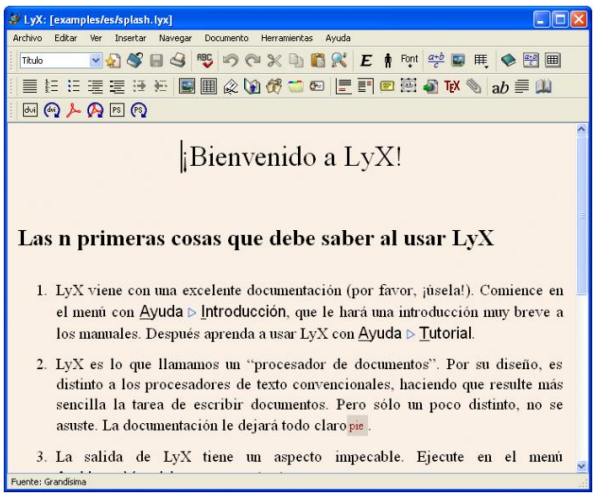


Figura 2.21: Interfaz LYX

2. **Kicad**[21]**:**
   Kicad (Figura 2.22) es un software Open Source, con una interfaz muy parecida a Eagle. Permite el diseño de esquemáticos, de footprints y la visualización 3D de los componentes. Fue creado por la Universidad de Grenoble en 1992, pero ahora funciona a través de una comunidad de desarrolladores. Además, el CERN[22] ha decidido firmar un acuerdo de colaboración con Kicad para mejorar el desarrollo del software para conseguir una potente herramienta de diseño de PCBs de gran complejidad completamente gratis.

El entorno Kicad está constituido por cinco partes:

Figura 2.22: Logo Kicad

1. Kicad (Figura 2.23): El administrador de proyectos.

[21] Información extraída de `http://www.kicad-pcb.org/display/KICAD/KiCad+EDA+Software+Suite`
[22] Información extraída de `http://www.ohwr.org/projects/cern-kicad/wiki/WorkPackages`

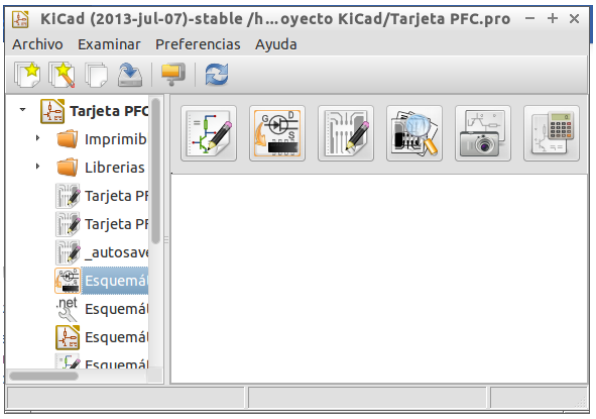


Figura 2.23: Interfaz Kicad

2. Eeschema (Figura 2.24): Realización de esquemáticos y creación de simbolos electrónicos. Tiene la opcion de importar y exportar netlist de otros programas como Eagle u Orcad.

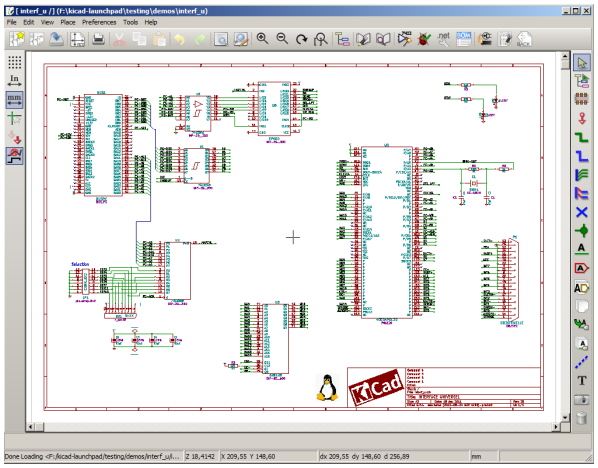


Figura 2.24: Interfaz Eeschema

3. Cvpcb (Figura 2.25): Selecciona los footprint de los componentes de nuestro esquemático.

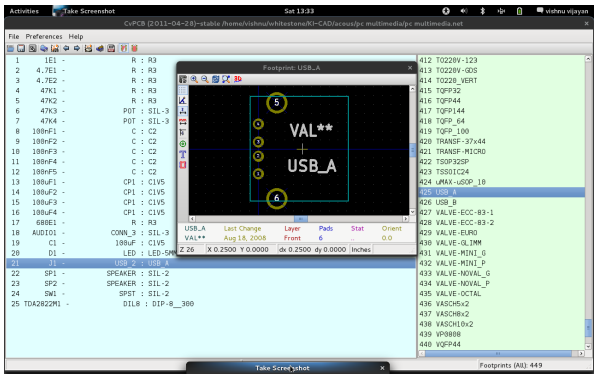


Figura 2.25: Interfaz Cvpcb

4. Pcbnew (Figura 2.26): Entorno de diseño de circuitos impresos. Cuenta con visualización 3D (Figura 2.27) y un editor de módulos(footprint) para realizar nuevos componentes. También cuenta con la opción de Importar/Exportar modelos CAD en .DXF. Esto nos ayudará a realizar PCBs de una geometría más compleja.

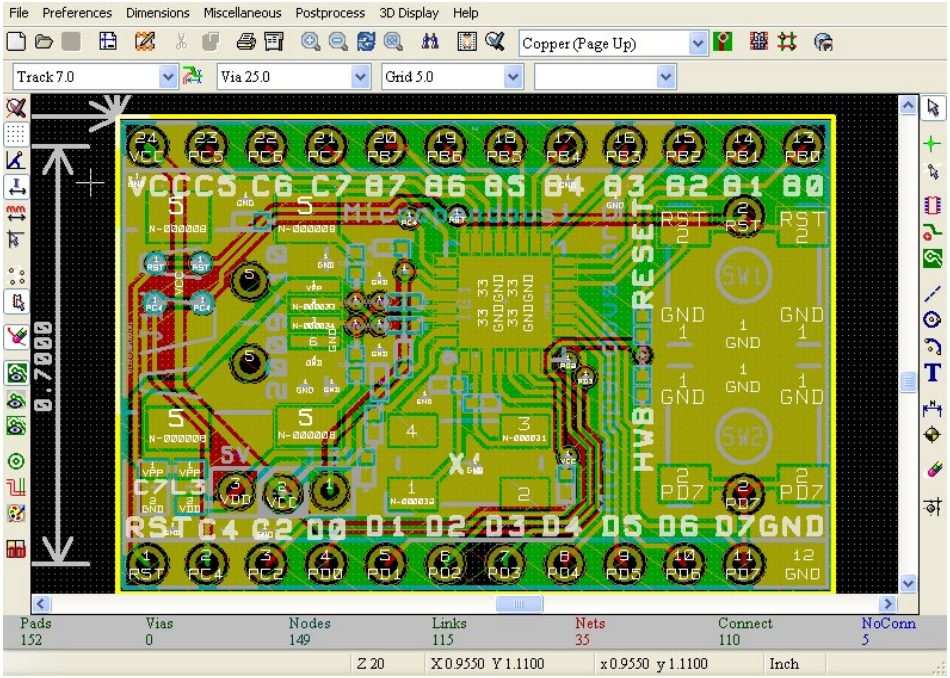


Figura 2.26: Interfaz Pcbnew

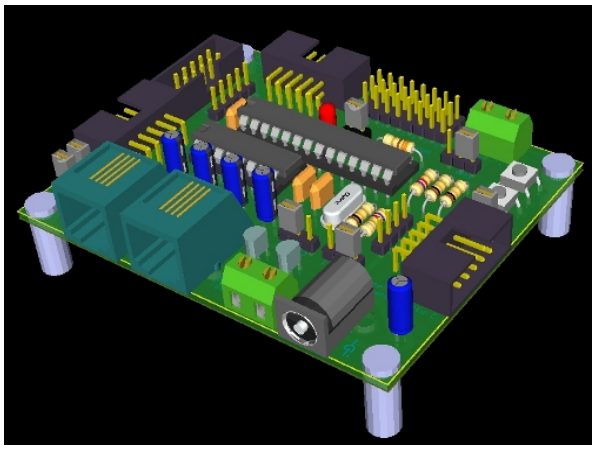


Figura 2.27: Renderizado 3D SKYPIC con Kicad

5. Gerber: Visualizador de ficheros Gerber.

Capítulo 3

Hardware

3.1. Introducción

En este capítulo desarrollaremos todo lo relacionado con el hardware que hemos utilizado en este proyecto.

Empezaremos explicando los distintos elementos que conformaban la maqueta, haciendo especial énfasis en los actuadores y el sensor. Después explicaremos los distintos elementos que hemos cambiado y añadido y el porqué de estos cambios.

Por último explicaremos toda la electrónica de acondicionamiento y control; y el proceso de diseño y realización de la placa fotosensible utilizada para hacer nuestra tarjeta de adquisición de datos.

3.2. Descripción de la maqueta.

3.2.1. Descripción general.

Nuestra maqueta (Figuras 3.1 y 3.2) está conformada por una estructura de metacrilato de 500x300x300mm y un suelo artificial de terrarios. Cuenta también con un módulo independiente (Figura 3.3) anexo de 120x120x120mm para calentar el aire con una bombilla calefactora e introducirlo con un ventilador. Otro ventilador se encarga de introducir aire del exterior de forma controlada mientras que una trampilla en la parte superior introducirá aire como perturbación en nuestro sistema.

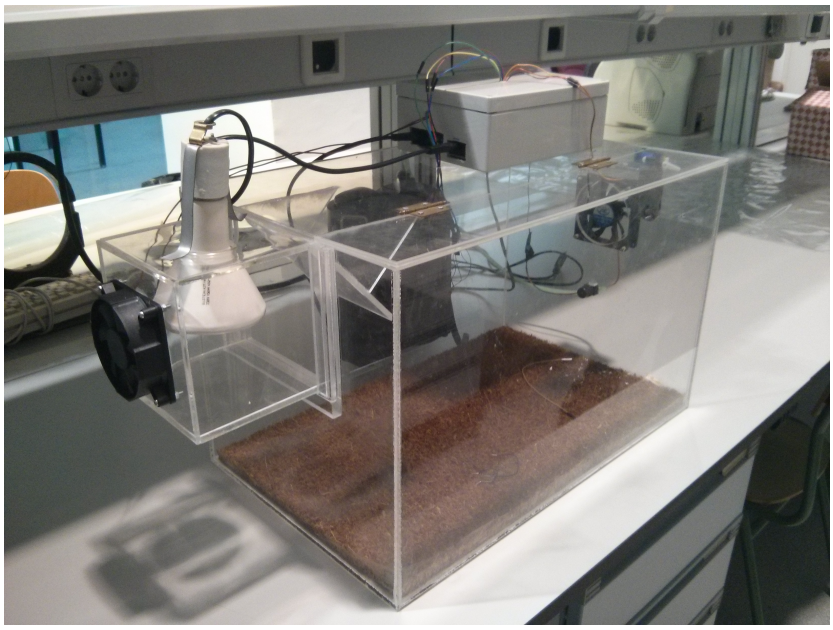


Figura 3.1: Foto de la maqueta

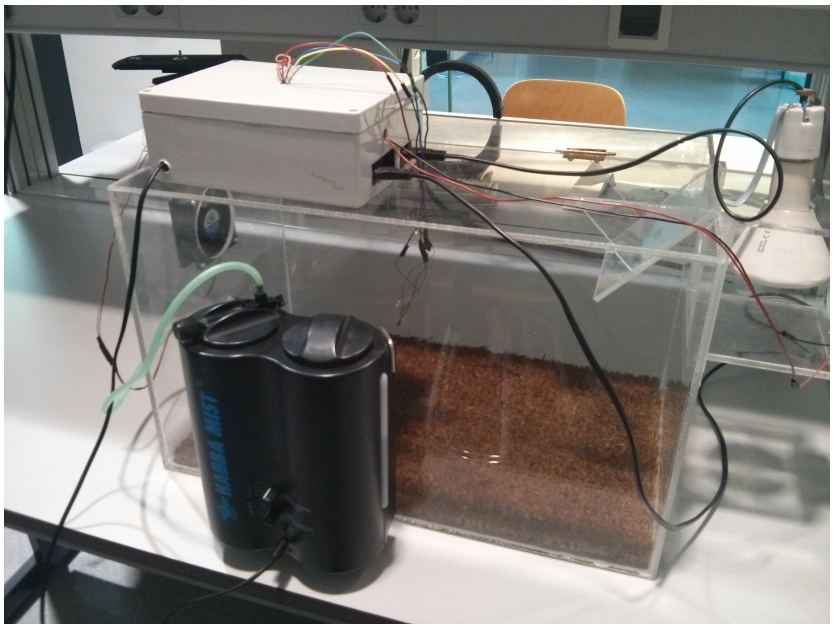


Figura 3.2: Foto maqueta.

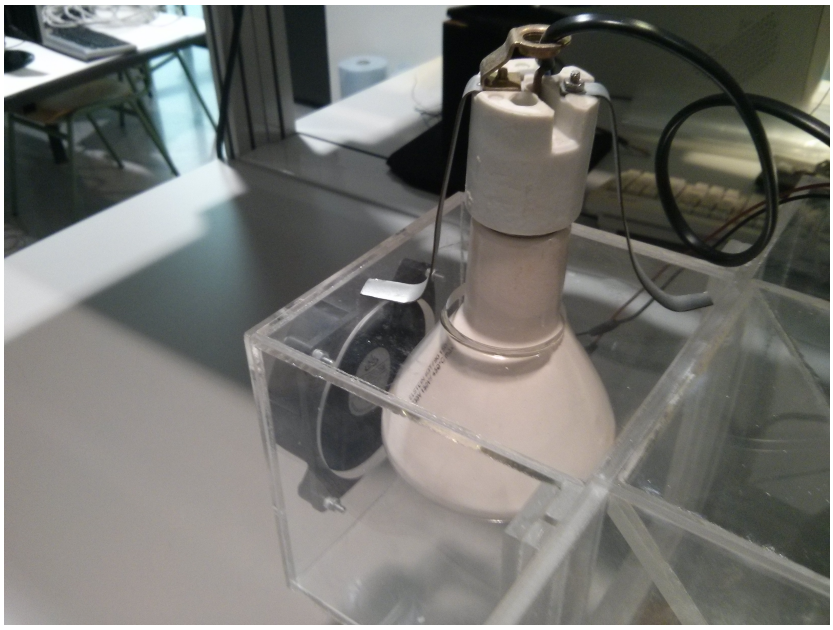


Figura 3.3: Módulo independiente

3.3. Actuadores.

Nuestra maqueta consta de los siguientes actuadores.

1. **Ventilador PC 12V (Figura 3.4):**
   Dos ventiladores controlados por PWM. Uno se encargará de introducir aire caliente al módulo central de la maqueta y el otro introducirá aire del exterior para reducir la temperatura. El control PWM nos permitirá regular la velocidad de estos, consiguiendo así conseguir el control deseado. Este control se puede realizar gracias a que Arduino Uno va equipado con 5 pines PWM para realizar el control.

Figura 3.4: Ventilador 12V

2. **Humedecedor Habba Mist:**
   Introduce agua vaporizada en nuestro sistema (Figura 3.5). Necesita 9V y 500mA de corriente,

lo controlaremos mediante PWM con un transistor. Cuenta con un dos temporizadores. Uno para regular el tiempo que dura el ciclo de funcionamiento, y el segundo sirve para regular el tiempo que transcurre entre ciclos. El máximo tiempo que puede funcionar es de 60 segundos consecutivos.

Figura 3.5: Humedecedor

3. **Bombilla calefactora (Figura 3.6):**
   Situada en el módulo independiente, será la encargada de calentar el aire dentro del módulo. Puede alcanzar 250ºC en la superficie, lo que es peligroso para la maqueta, así que se optará por utilizarla a mitad de su potencia. Se controlará con un Triac y un Optotriac, para aislar la parte de continua de la parte de alterna.

Figura 3.6: Bombilla calefactora

3.4. Sensores.

La maqueta llevaba incorporada el SHT75 (Figura 3.7) de Sensirion. Un sensor digital de temperatura y humedad que cuenta con un protocolo de comunicación propio, muy parecido al I2C. Esto nos

ahorra electrónica de acondicionamiento, ya que el sensor es capaz de sacar los valores sin necesidad de electrónica añadida. Arduino posee una librería desactualizada para la comunicación entre el sensor y Arduino. Más adelante explicaremos como arreglarlo.

Al escoger este sensor, no podremos utilizar los sketchs de control que hay de Scilab a Arduino. Tendremos que diseñar un sketch que nos lea los valores del sensor y los envíe al puerto serie, y realizar un script en Scilab que nos lea los valores del puerto serie.

En los datasheet del sensor aparecen las funciones de comunicación programadas en C por si se utiliza otro hardware de control como los PIC.

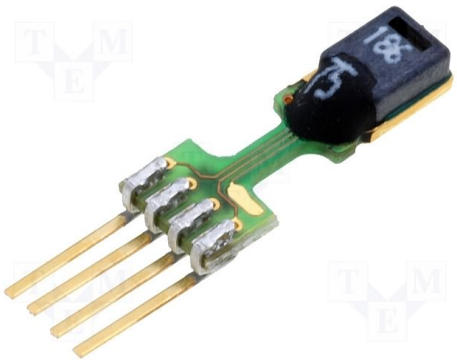


Figura 3.7: Sensor SHT75

3.5. Cambios realizados en la maqueta.

Para renovar la electrónica de la maqueta decidimos cambiar el PIC18F255 (Figura 3.8) por Arduino Uno. Este dispositivo nos proporciona un entorno de programación mucho más intuitivo y sencillo que el PIC.

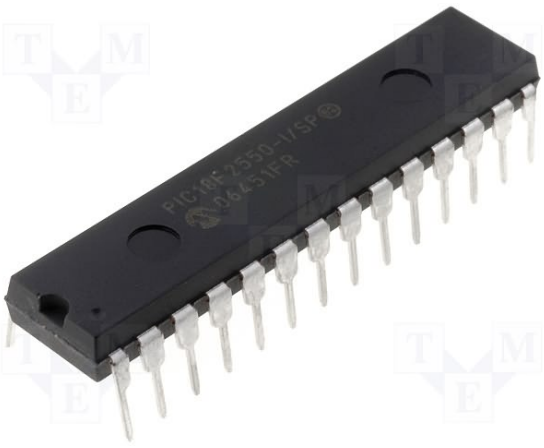


Figura 3.8: PIC 18F255

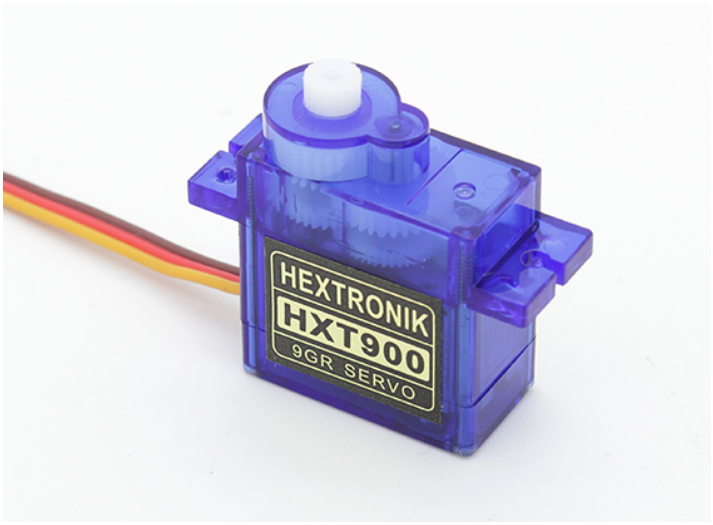


Figura 3.9: Servomotor

Arduino Uno también tiene la ventaja de que no necesita ninguna electrónica añadida como cristal de cuarzo o condensadores para la señal de reloj. Esto Arduino Uno lo lleva en su placa y no tenemos que preocuparnos por ello.

Tanto Arduino Uno como el PIC contienen pines de PWM para el manejo de los actuadores de nuestro sensor. La única diferencia sustancial entre el el PIC y el Arduino Uno constituye en la frecuencia de la señal de PWM. El PIC tiene la capacidad de oscilar hasta a 48KHz mientras que el PWM de Arduino no pasa de 500Hz. Esto se traduce en un algo ruido molesto en el giro de los ventiladores cuando no están a máxima potencia, pero que no influye en el funcionamiento del ventilador.

Otro cambio realizado ha sido la automatización de la trampilla. Esto se ha realizado con un pequeño servomotor (Figura 3.9) y un soporte para ajustarlo en la parte interior de la maqueta(Figuras 3.10 y 3.11). Arduino Uno posee una librería llamada "Servo.h" con la que se puede manejar este tipo de servos introduciendo el ángulo que nosotros elijamos.

Para llevar a cabo todos estos cambios debemos diseñar una tarjeta de adquisición nueva con los componentes añadidos.

3.6. Desarrollo de la tarjeta de adquisición.

3.6.1. Electrónica.

En este apartado vamos a explicar los distintos circuitos electrónicos necesarios para el control de nuestra maqueta. La circuitería está dividida en varias etapas: alimentación, control y acondicionamiento de los ventiladores, control y acondicionamiento de la bombilla.

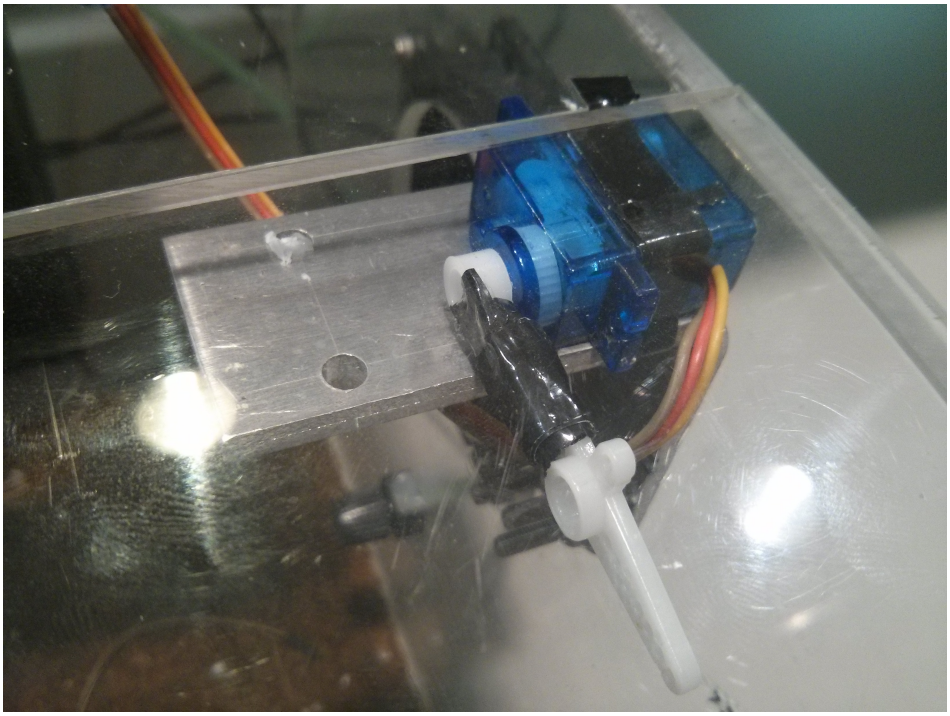


Figura 3.10: Colocación Servomotor.

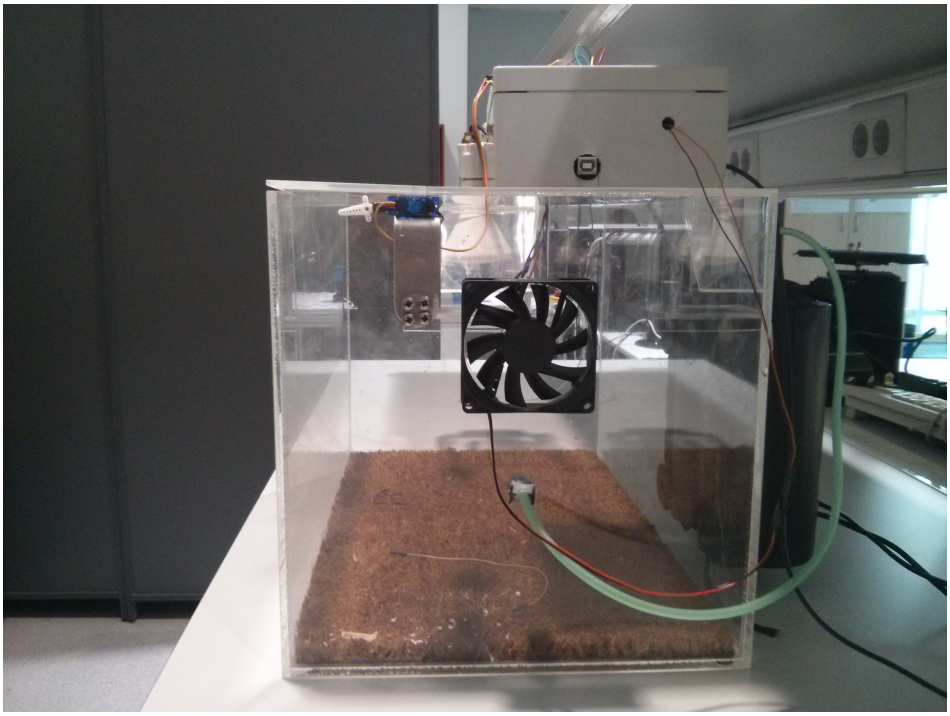


Figura 3.11: Atornillado Servomotor.

3.6.1.1. Alimentación:

La etapa de alimentación consta de la rectificación de la señal de 230V alterna a 12V y 5V de continua. La etapa de alimentación cuenta de lo siguiente:

- **Transformador 230V a 12V(Fig 3.12):**
  Con este transformador conseguiremos tener 12V en el secundario. Acompañando al transformador colocaremos dos condensadores de 4700µF para conseguir una señal con un rizado muy pequeño.

Figura 3.12: Transformador

- **Reguladores de tensión(Fig 3.13):**
  Dos reguladores, un 7805 y un 7812. Con estos dos reguladores conseguiremos las señales de 5V y 12V que necesitamos. Además necesitaremos otros dos condensadores de 470µF y 100nF en cada regulador. Las señales de 5V y 12 V se utilizarán para:

1. **5V**: Esta señal se utilizará para la alimentación del servomotor, alimentar el sensor de temperatura/humedad y un optoacoplador para el manejo de la bombilla.

2. **12V:** Esta señal se utilizará para la alimentación de los ventiladores y del humedecedor.

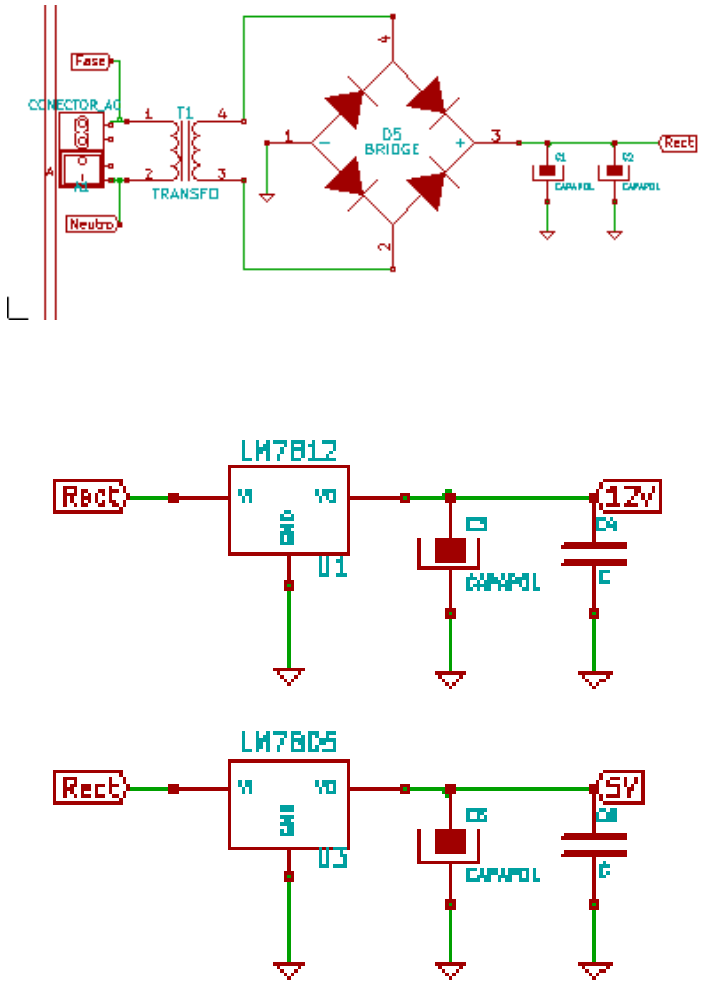


Figura 3.13: Esquemático Etapa Alimentación

3.6.1.2. Acondicionamiento y control de los ventiladores.

Esta etapa (Figura 3.14) consta de los actuadores que irá controlados por PWM. Estos son: la bombilla, el ventilador caliente, el ventilador frío y el humedecedor.

Ya que el control directo por PWM es imposible, necesitamos un circuito conmutador para conseguir que la señal PWM de Arduino maneje el circuito. Para esto utilizaremos lo siguiente:

- **Transistor BD135:**
  Es un modelo de transistor utilizado especialmente para conmutaciones, ya que es bastante rápido. La señal de PWM de Arduino entrará por la base del BD135 y mandará conmutar la tensión que hay entre colector y emisor, esto hará conseguir una amplificación de la señal de PWM de 5V a 12V. La base necesita una resistencia para que circule una intensidad para activar el transistor.

- **Diodo 1N4007:**
  Con ellos se pretende hacer circular la energía acumulada en la bobina del ventilador después del corte y así evitar que sea destruido el transistor por una corriente inversa.

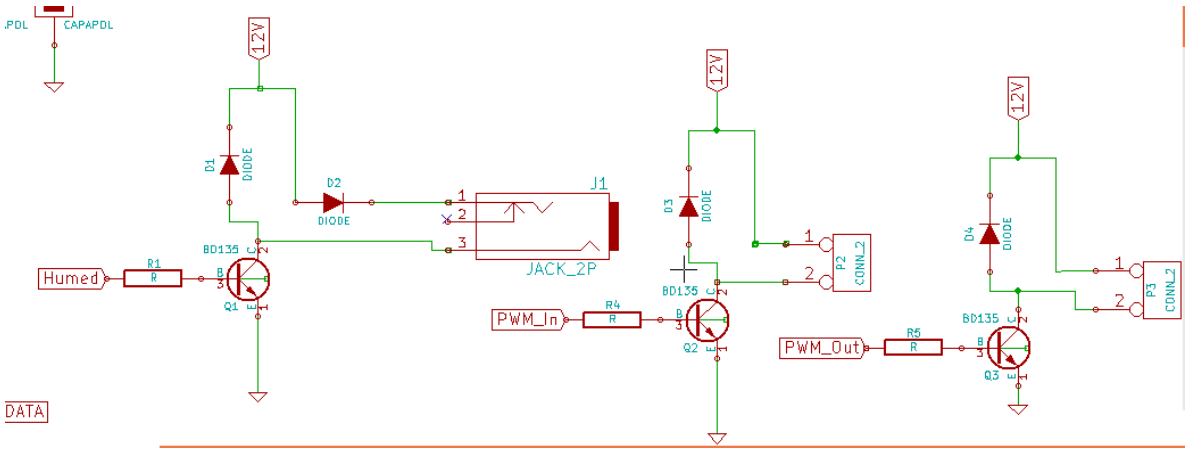


Figura 3.14: Etapa control actuadores

3.6.1.3. Acondicionamiento y control de bombilla.

Para controlar la bombilla utilizaremos un triac y un optoacoplador, así conseguiremos controlarla mediante la señal PWM de Arduino. La filosofía del circuito consiste en utilizar un optoacoplador de paso por cero que habilite el paso de la tensión alterna por la bombilla. Así, cuando la señal PWM pase por cero, habilitará el paso de corriente alterna por la bombilla calefactora, consiguiendo así conmutar la señal alterna por PWM (Figura 3.15). A continuación haremos una descripción detallada de los componentes.

- **MOC3041:**
  El componente posee dos entradas y dos salidas. Una de las entradas será para fijar la tensión de 5V, la otra se utilizará para mandar la señal PWM de Arduino. Entre los dos terminales se encuentra un diodo emisor de luz que se activa cuando la patilla de PWM de Arduino pasa a 0V. Además de utilizarse para conmutar, el MOC3041 es muy útil para aislar la señal de continua de Arduino con la señal de alterna introducida a la bombilla calefactora.

- **Triac:**
  Este utilizará la señal mandada por el MOC3041 para conmutar la señal alterna que irá a la bombilla.

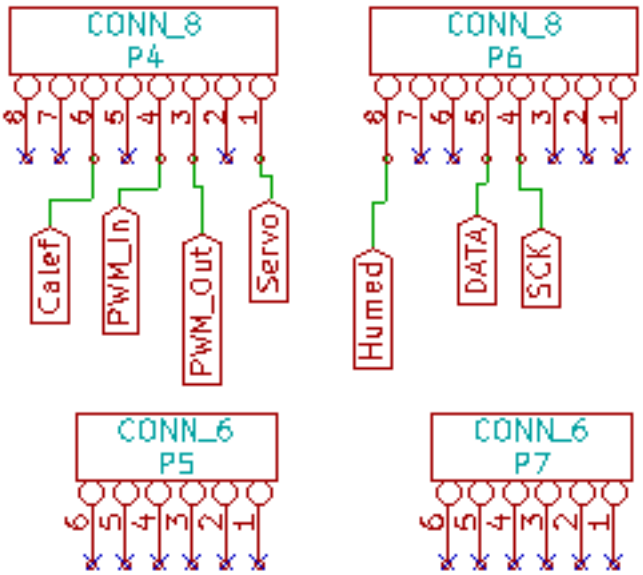


Figura 3.16: Pines Arduino

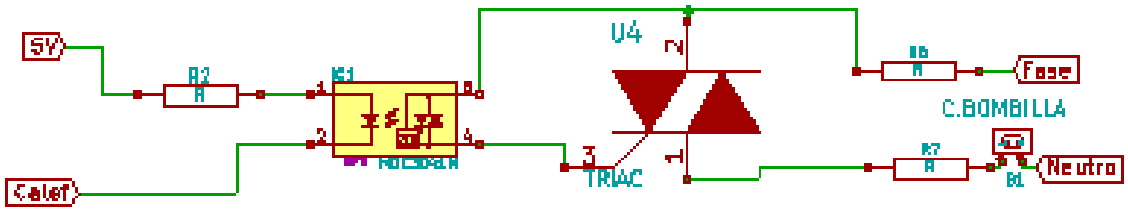


Figura 3.15: Etapa control bombilla

3.6.1.4. Salidas de Arduino.

Para colocar Arduino en nuestra placa utilizaremos arrays de pines (Figura 3.16) colocados a la misma distancia, lo que nos permitirá acoplar Arduino a nuestra placa de forma sencilla.

3.7. Realización de la PCB.

El procedimiento para la realización de la PCB se divide en tres partes.

1. **Isolado:**
   Consiste en radiar con luz, esta vez un armario con un tubo fluorescente (Figura 3.17), para que se queden grabadas la forma de nuestro acetato (Figuras 3.18 y 3.19) la cara de las pistas y la cara con la forma de los componentes . Esto se realizará durante dos minutos.

Figura 3.17: Armario fluorescente

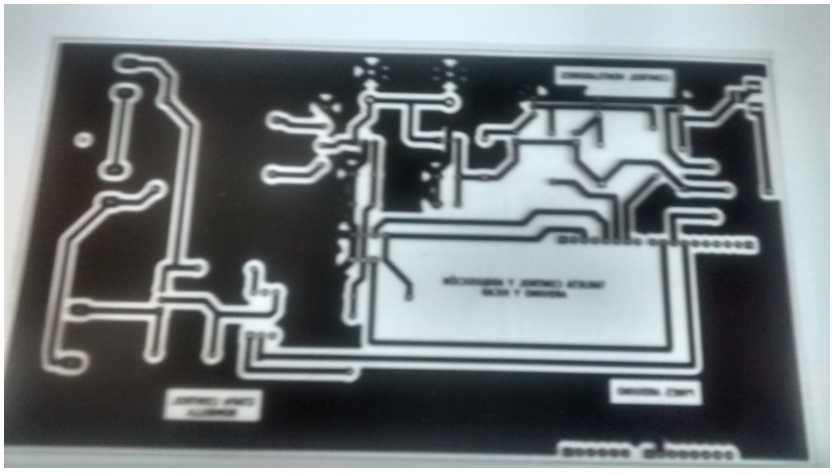


Figura 3.18: Imagen acetato capa de GND

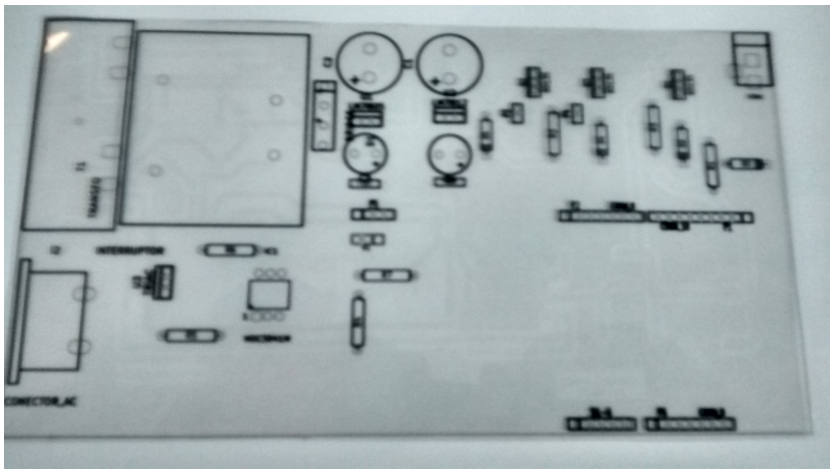


Figura 3.19: Acetato cara de componentes

2. **Revelado:**
   Esta parte consiste revelar el cobre con la forma de nuestro acetato. Utilizaremos una mezcla de agua y sosa cáustica en una bandeja (Figura 3.20). La proporción suele ser un tapón de sosa caustica por cada 2 litros de agua. Dejaremos revelar el tiempo suficiente para que se aprecien las pistas y el color del cobre sea brillante (Figura 3.21). Si dejamos la placa más tiempo, la sosa acabará borrando las pistas de nuestro circuito.

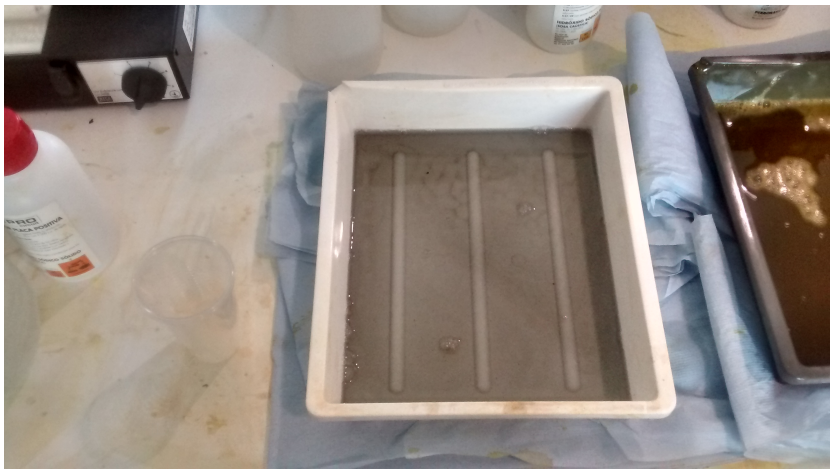


Figura 3.20: Bandeja para el revelado

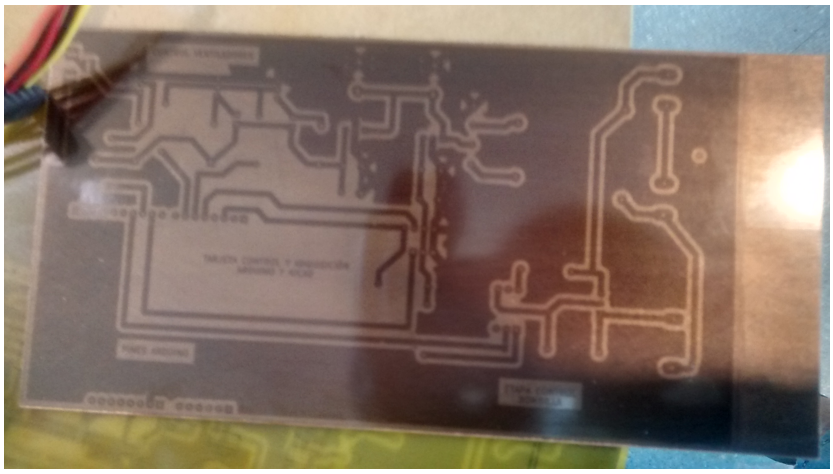


Figura 3.21: Placa revelada

3. **Atacado:**
   Consiste en eliminar el cobre sobrante con una mezcla de agua y cloruro férrico. Esta etapa es delicada y se debe de medir muy bien la mezcla necesaria para acabar con el cobre sobrante y no con todas las pistas. Se necesitan guantes y protección para boca, ojos y nariz, ya que las sustancias químicas utilizadas son corrosivas. Una vez realizado el atacado, procedemos a limpiar la placa de restos de cloruro férrico, comprobamos la continuidad de las pistas, y por último, soldamos los componentes a nuestra placa (Figura 3.22).

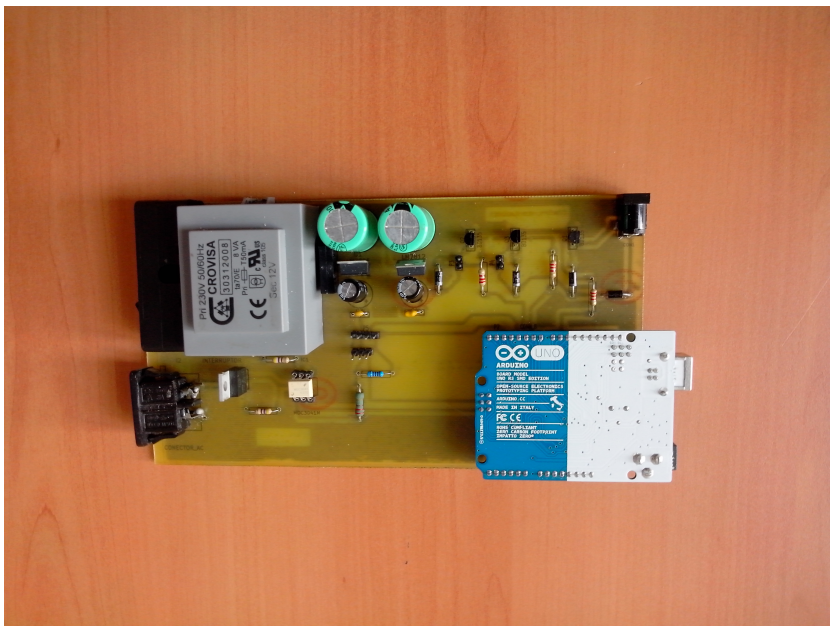


Figura 3.22: Placa final

Capítulo 4

Desarrollo del software

4.1. Introducción.

En este capítulo presentaremos todo el software que hemos desarrollado para la realización de nuestro proyecto. Hemos dividido este apartado en varias secciones diferentes dependiendo del problema a resolver, y nos ha quedado así:

1. **Comunicación:**
   En esta sección explicaremos todo el software desarrollado relacionado con la comunicación de nuestra maqueta: la comunicación entre Arduino y el sensor mediante la librería "Sensirion.h", la comunicación entre Scilab y Arduino mediante el ATOM de Scilab "Serial communication Toolbox".

2. **Identificación:**
   Aquí ilustraremos el proceso para el diseño de la GUI utilizada para la recogida de datos y la identificación de nuestro sistema. Para ello hemos utilizado la herramienta "GUI builder" de Scilab.

3. **Control:**
   Daremos un recorrido por las funciones principales de Xcos que hemos utilizado para el control de nuestro sistema y explicaremos las principales diferencias que se encuentran entre Xcos y Simulink.

4.2. Comunicación.

4.2.1. Comunicación SHT75 y Arduino.

Para la comunicación entre el Arduino Uno y el SHT75 descargaremos la versión 2.0 de la librería "Sensirion", que nos proporciona funciones de lectura de valores y comprobación de redundancia cíclica(CRC), además de códigos de ejemplo para comprobar su funcionamiento.

Uno de los problemas del uso de esta librería es que se limita el uso a las versiones del IDE de Arduino menores que 1.X.X. Debido a que a partir de Arduino 1.0.X se modificaron y se unieron varias librerías, al cargar la librería Sensirion en las nuevas versiones dará multitud de errores de compilación, como vemos en la figura siguiente (4.1).

Figura 4.1: Error compilando librería Sensirion

Para subsanar este error debemos editar el archivo "Sensirion.cpp" de la librería descargada. Lo único que tenemos que hacer es cambiar varias sentencias en la declaración de las librerías al inicio del código.

Cabecera inicial (Figura 4.2):

```
extern "C"
{ // AVR LibC Includes #include <stddef.h>
#include <stdint.h>
#include <math.h>
// Wiring Core Includes #include "WConstants.h" }
#include "Sensirion.h"
```

Figura 4.2: Cabecera inicial.

Como hemos dicho anteriormente, con la actualización de del IDE de Arduino algunas librerías se incluyeron en una sola, es el caso de "Wconstans.h", que ahora está dentro de la nueva librería "Arduino.h".

Cabecera editada (Figura 4.3):

```
extern "C" { // AVR LibC Includes
#include <stddef.h>
#include <stdint.h>
#include <math.h>
}
// Wiring Core Includes
#include "Sensirion.h"
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
```

Figura 4.3: Cabecera Editada.

Con la estructura if-else conseguimos que dependiendo de la versión de nuestro IDE el programa cargue "Arduino.h" o "Wprogram.h". Con esta edición la librería no debe dar problemas en cualquier edición de Arduino IDE. Procederemos ahora a mostrar un ejemplo de comunicación entre el sensor y el Arduino.

Con el sensor conectado a nuestro Arduino, cargaremos uno de los ejemplos que viene implementado en la librería Sensirion, se llama "SimpleSensirion" (Figura 4.4). Este programa consiste en la lectura y envío por puerto serie de los valores de temperatura, humedad y punto de rocío del ambiente. El código es el siguiente:

```
#include <Sensirion.h>
const uint8_t dataPin = 2; //Declaracion patillas arduino
const uint8_t clockPin = 3;
float temperature; //Declaracion variables
float humidity;
float dewpoint;
Sensirion tempSensor = Sensirion(dataPin, clockPin); //Declaracion objeto sensor
void setup()
{
Serial.begin(9600); //Declaracion puerto serie
}
void loop() {
tempSensor.measure(&temperature, &humidity, &dewpoint); //Llamada sensor
Serial.print("Temperature: "); //Escribe texto para acompañar al valor
Serial.print(temperature); //Escribe valor en la misma linea
Serial.print(" C, Humidity: ");//Escribe texto para acompañar al valor
Serial.print(humidity); //Escribe valor en la misma linea
Serial.print(" %, Dewpoint: ");//Escribe texto para acompañar al valor
Serial.print(dewpoint); // Escribe valor en la misma linea
Serial.println(" C");
delay(5000); //Tiempo de espera para la siguiente ejecucion
}
```

Figura 4.4: Código SimpleSensirion

Ahora utilizaremos el visor de puerto serie del entorno de programación de Arduino (Figura 4.5) para ver los resultados:

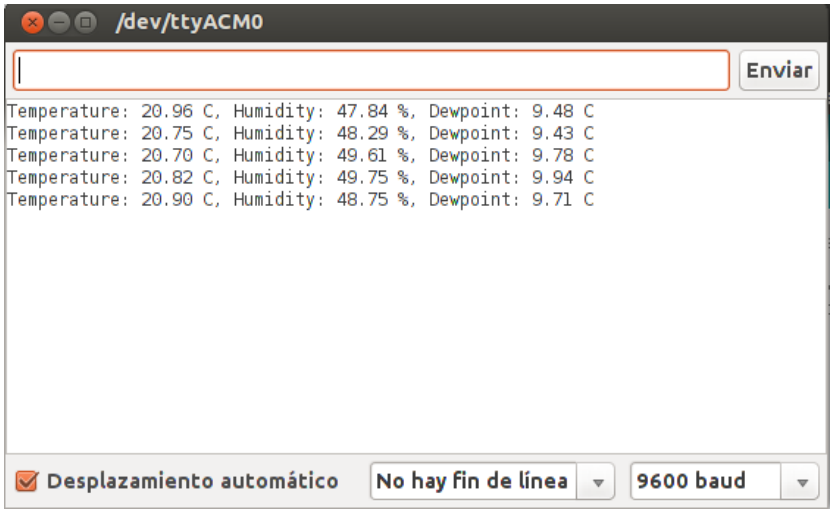


Figura 4.5: Visor puerto serie

Como podemos observar en la figura (4.5), las funciones de la librería consiguen interpretar los valores emitidos por el sensor.

4.2.2. Comunicación Arduino-Scilab.

4.2.2.1. Comunicación Arduino-Interfaz Scilab.

Para realizar la comunicación Arduino-Scilab hemos decidido hacerlo mediante USB por emulación de puerto serie. Para tener las funciones de lectura y escritura de puerto serie en Scilab necesitamos descargarnos antes un ATOM o paquete extra que tiene Scilab en sus repositorios. Para poder descargarlo necesitamos arrancar Scilab y pinchar con el ratón en la casilla de ATOMS  . Aquí entraremos en la interfaz del repositorio (Figura 4.6), donde podremos elegir los ATOMS necesarios para nuestro proyecto. Nosotros descargaremos el "Serial Comunication Toolbox".

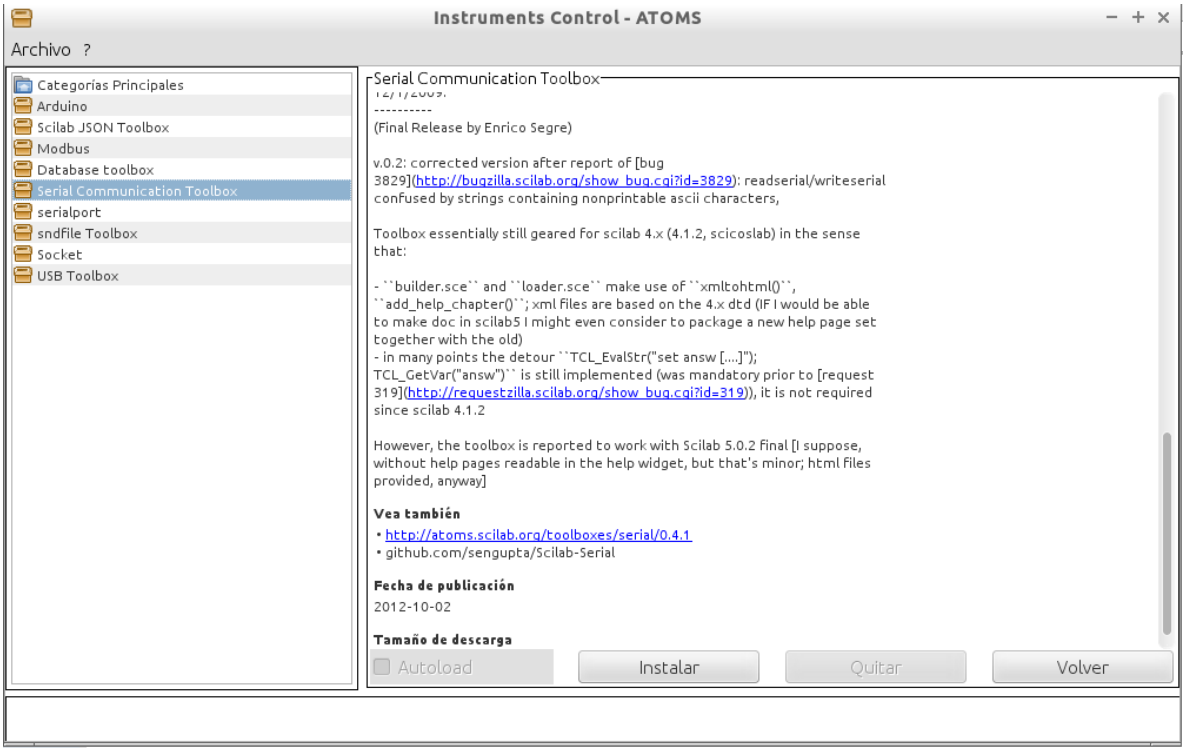


Figura 4.6: Instalación ATOMS

Este paquete contiene las funciones necesarias de comunicación que nosotros vamos a utilizar para comunicar Arduino y Scilab.

El paquete está compuesto por las siguientes funciones:

- **openserial:**
  Esta función nos abre el puerto de comunicación serie en el que tengamos conectado nuestro Arduino. En esta función se declara también los bits de parada, la velocidad de transmisión, etc. Hay que tener en cuenta que la declaración del puerto serie es distinta en Windows que en GNU/Linux.

```
h=openserial(1,"9600,n,8,1") // Declaración puerto serie.
writeserial(h,"#02"+ascii(13)) //Escritura puerto serie.
xpause(200000)  // Comando pausa de dos segundos.
readserial(h)  // lectura de puerto serie.
closeserial(h) // cierre de puerto serie.
```

Figura 4.7: Ejemplo comunicación serie

- **readserial:**
  Función para leer lo enviado por el puerto serie. En nuestro caso lo utilizaremos para leer los datos de temperatura y humedad que tengamos. Con esta función podremos elegir el numero de bytes a leer.

- **writeserial:**
  Con esta función enviaremos los valores de los actuadores por el puerto serie para que Arduino los lea.

- **closeserial:**
  Esta función se utiliza para cerrar el puerto serie y dar por finalizada la comunicación.

A continuación (Figura 4.7) podemos ver un ejemplo sencillo de comunicación serie con Scilab en Windows.

4.2.2.2. Sketch de Arduino.

El sketch necesario para la comunicación con Scilab ha sido construido a partir del ejemplo "SeriallCallResponse". Esta estructura consiste en la realización de dos funciones; una a realizar mientras Arduino espera a recibir un valor por el puerto serie, en nuestro caso una variable llamada "header". Y en caso de recibir esa variable, activar la lectura y escritura tanto del sensor como de los actuadores. Dependiendo del valor del header, Arduino irá a un bucle if diferente. Hemos dividido el software en tres bloques if diferentes, uno para enviar temperatura, otro para enviar humedad y el tercero para enviar ambas variables.

```
#include<Servo.h>
#include<Sensirion.h>
int header=0;
const uint8_t ventc=6; //pines arduino
const uint8_t ventf=5;
const uint8_t bombilla=3;
const uint8_t humed=11;
const uint8_t servopin=12;

const uint8_t datos=10; //pines sensor
```

```
const uint8_t clock=9;
float temperatura;  //variables del sistema
float humedad;
float dewpoint;
float servo1ang;
int pwmc;
int pwmf;
int pwm_bomb;
int pwm_hum;
Sensirion sensor(datos,clock);
Servo myservo1;
void setup() {
Serial.begin(9600);
myservo1.attach(servopin);  //Declaracion pines servos
pinMode(3,OUTPUT);
pinMode(5,OUTPUT);
pinMode(6,OUTPUT);
pinMode(11,OUTPUT);
establishContact(); //funcion espera. Espera la llamada de Scilab para ejecutar
//alguna de las funciones de arduino.
}
void loop() {
if (Serial.available()>0){ //Solo funciona cuando el valor de puerto serie es mayor que 0 header=Seri
if(header==1){//envio humedad y lectura
sensor.measure(&temperatura,&humedad,&dewpoint); //llamada a sensor
pwmc=Serial.read(); //lectura valores de los actuadores
pwmf=Serial.read();
servo1ang=Serial.read();
Serial.print(humedad);
Serial.print(" ");
analogWrite(ventf,pwmf); //escribir valores de los actuadores
analogWrite(ventc,pwmc);
analogWrite(bombilla,128);
myservo1.write(servo1ang);
}
else if(header==2){
sensor.measure(&temperatura,&humedad,&dewpoint);
pwmc=Serial.read();
pwmf=Serial.read();
servo1ang=Serial.read();
```

```
Serial.print(temperatura);
Serial.print(" ");
analogWrite(ventf,pwmf);
analogWrite(ventc,pwmc);
analogWrite(bombilla,128);
myservo1.write(servo1ang);
}
else if(header==3){
sensor.measure(&temperatura,&humedad,&dewpoint);
pwmc=Serial.read();
pwmf=Serial.read();
//servo1ang=Serial.read();
Serial.print(temperatura);
Serial.print(" ");
Serial.print(humedad);
Serial.print(" ");
analogWrite(ventf,pwmf);
analogWrite(ventc,pwmc);
analogWrite(bombilla,128);
myservo1.write(servo1ang);
}
}
}


void establishContact() {
while (Serial.available() <= 0) {
myservo1.write(0);
delay(10);
}
}
```

Figura 4.8: Sketch Comunicación Arduino-Scilab

4.3. Identificación.

Para esta etapa decidimos implementar una interfaz de usuario con la que comunicar la maqueta. Para esto descargamos el ATOM "guibuilder" (Figura 4.9). Este paquete está disponible en el repositorio de Scilab. Aunque es una aplicación algo modesta, cumple a la perfección con lo que nosotros necesitamos.

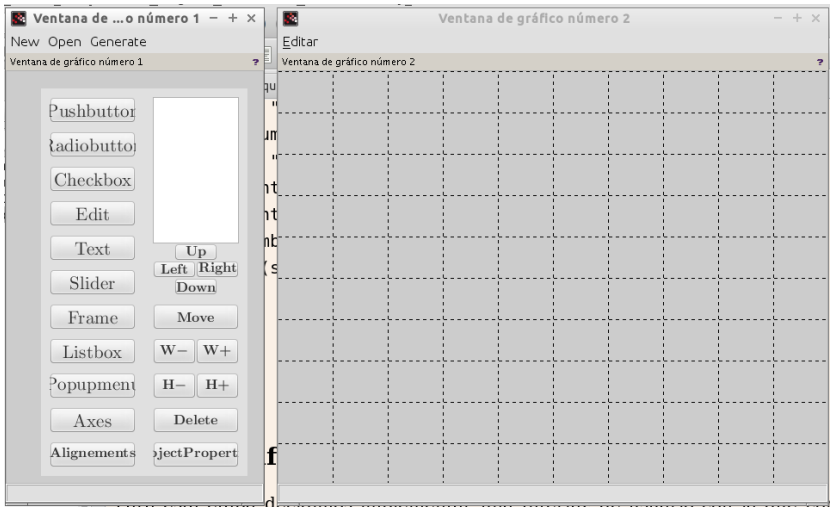


Figura 4.9: Guibuilder

Como podemos ver en la figura (4.9), con "guibuilder" es bastante sencillo realizar ejes, botones, barras deslizantes y otros elementos en nuestro espacio de trabajo. Nuestra interfaz contará con tres ejes y con una serie de botones para controlar la comunicación del sistema (Figura 4.10).

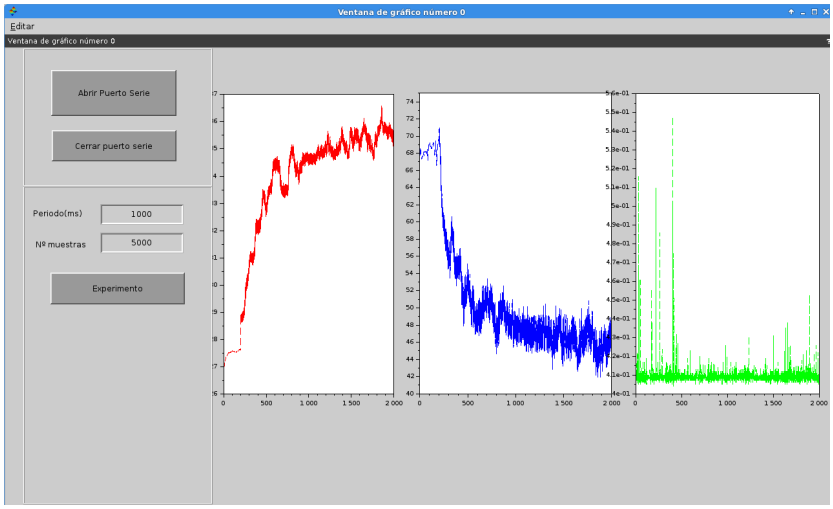


Figura 4.10: GUI experimentos

Los tres ejes se utilizarán para visualizar la temperatura, la humedad y el tiempo entre ciclos. Esto se utilizará para comprobar si nuestro sistema trabaja a tiempo real.

Los botones superiores se utilizarán para abrir o cerrar la comunicación del puerto serie. Y por último, en los recuadros podremos escribir el periodo de muestreo y la cantidad de muestras necesarias para realizar nuestros experimentos.

La programación de las señales de entrada no se pueden programar mediante el GUI. Decidimos

programarlas desde el script de Scilab porque el "guibuilder" no tiene herramientas suficientes para programar los distintos escalones que introduciremos a la señal en nuestro experimento. El código completo del GUI se adjuntará en un apéndice.

4.4. Control.

Para desarrollar el software del control de nuestra maqueta, hemos utilizado el entorno Xcos de Scilab. Xcos es un entorno de programación por bloques muy similar a Simulink de Matlab.

4.4.1. Entorno Xcos

Xcos[1] es un editor gráfico para la construcción de modelos de sistemas dinámicos. Los modelos se pueden construir, cargar, guardar, compilar, simular, mediante la GUI de Xcos. El entorno Xcos presenta los siguientes elementos:

Editor (Figura 4.11):
El editor permite el diseño de diagramas de flujo que representan a un sistema dinámico de bloques definidos en las paletas. Los usuarios pueden personalizar los bloques estándar y definir otros nuevos.

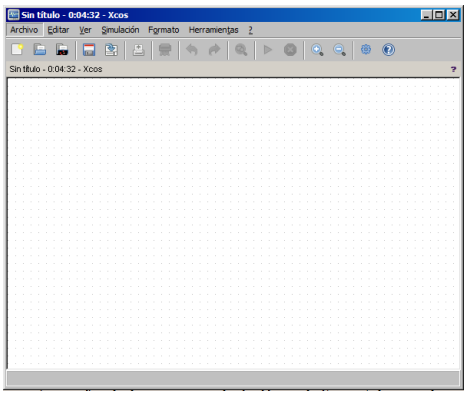


Figura 4.11: Editor Xcos

Explorador de paletas (Figura 4.12):
El explorador de paletas contiene una lista donde se enumeran todos los bloques de Xcos estándar agrupados por categorías (procesamiento de señal, eléctrica, hidráulica, derivada, integral, etc.)

Compilador / Simulador:
El entorno de Xcos permite, a partir del modelo creado, la simulación de sistemas. Los datos resultantes de la simulación gráfica se pueden ver en tiempo real, utilizado en Scilab para el procesamiento posterior. Es importante tener bastante claro la configuración del simulador para obtener los resultados que necesitamos. En la ventana integrador (Figura 4.13) aparecen los siguientes parámetros:

[1] Información extraída de `http://cacheme.org/course/programacion-en-el-entorno-xcos-scilab/`

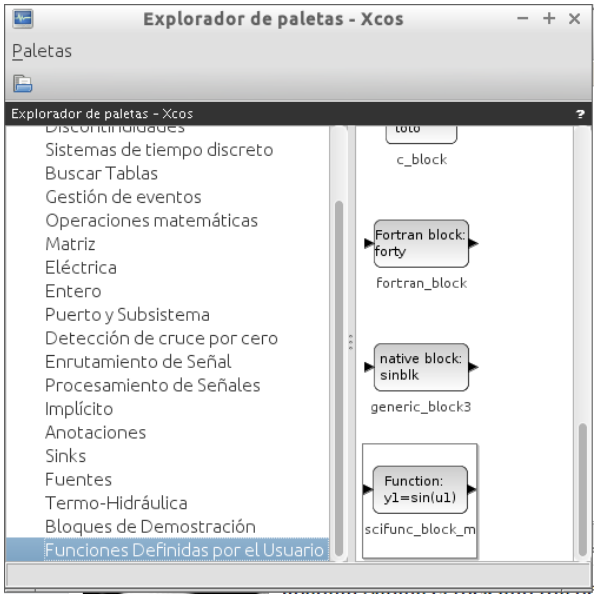


Figura 4.12: Explorador de paletas

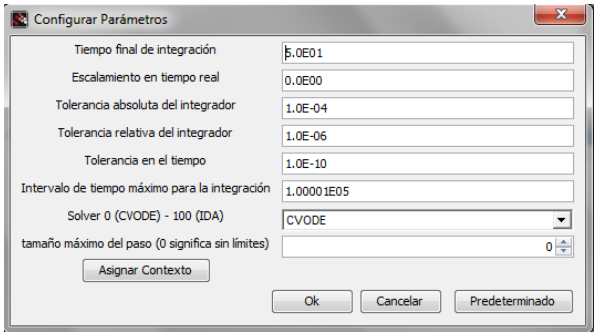


Figura 4.13: Menú Integrador

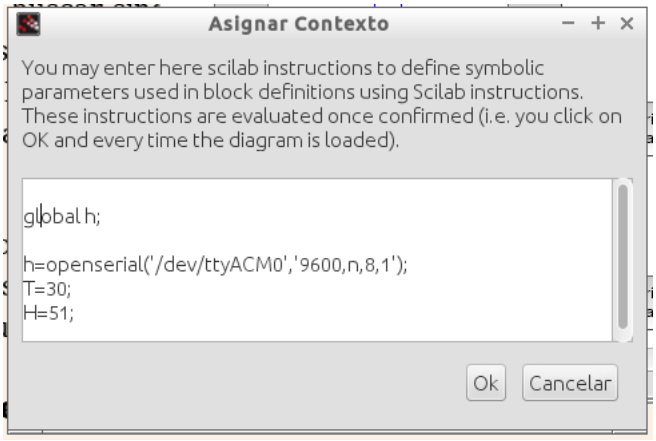


Figura 4.14: Ventana de Asignar Contexto

- Tiempo final de integración: la integración (simulación) termina en este momento, siempre se empieza desde 0.
- Escalamiento en tiempo real: fuerza la simulación en tiempo real mediante el establecimiento de simulación en Xcos en una unidad de tiempo de 1 segundo.
- Tolerancias absolutos del integrador: propiedades Solver.
- La tolerancia relativa de tiempo: el intervalo de tiempo más pequeño para el cual se utiliza el solucionador de ecuaciones diferenciales ordinarias (ODE) que actualiza los estados continuos.
- Intervalo de tiempo máximo para la integración: el intervalo de tiempo máximo para cada llamada a solver.it debe ser reducido si se produce el mensaje de error "Demasiadas llamadas".
- Solver: seleccionar la solución numérica para ser utilizada. Tienes la posibilidad de elegir entre el ODE Solver y una solucionador de ecuaciones diferenciales algebraicas (AIF). Si Xcos detecta que el modelo requiere un solucionador de la AIF, el Xcos muestrará un cuadro de información sobre el cambio automático a este solucionador.
- Tamaño máximo de paso: el paso máximo de tiempo que tarda el solucionador. Este parámetro es útil para buscar singularidades localizadas en una respuesta del sistema monótono. Para este tipo de respuesta, la solución numérica aumenta automáticamente el tamaño de paso para disminuir el tiempo de cálculo. Si piensas que su sistema está en este caso, disminuye el valor de este parámetro.

Asignar contexto (Figura 4.14):

En esta opción declararemos todo lo relacionado con las condiciones iniciales de nuestro sistema. En el nuestro, declararemos la apertura del puerto serie y los valores iniciales de temperatura y humedad.

4.4.2. Bloques más destacados Xcos.

Esta sección la dedicaremos a hablar sobre los bloques más significativos que utilizaremos en la programación de nuestro sistema de control.

- **Clock_c (Figura 4.15):**
  A diferencia de Simulink, Xcos tiene una señal de reloj con la que se sincronizan todos los bloques que tienen entrada de evento. Con este bloque se programará el inicio del programa y el periodo de la simulación.

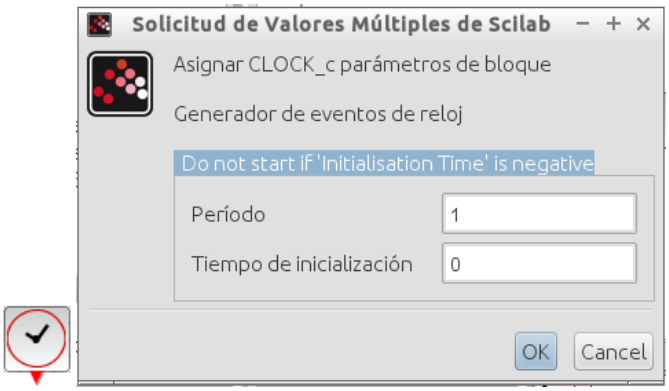


Figura 4.15: Icono y menú del bloque Clock_c

- **Variable delay (Figura 4.16):**
  Este bloque es necesario en cada sistema en tiempo real que utilicemos, ya que nos ayuda a eliminar el error "algebraic loop". Se configurará con un delay a 0 segundos y nuestro sistema funcionará perfectamente. Simulink ya tiene este error subsanado en las versiones recientes.

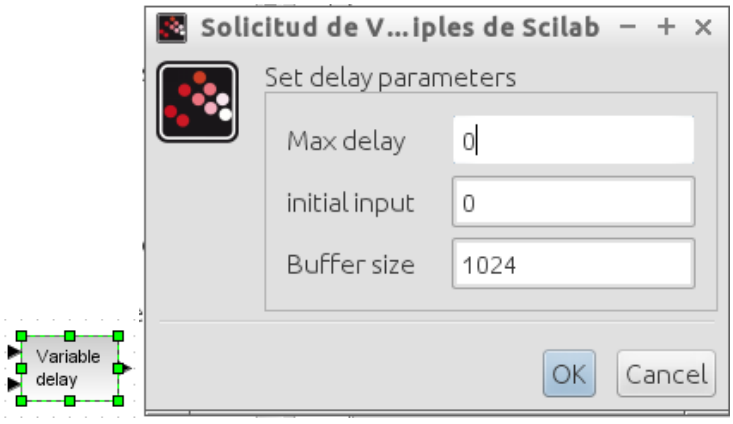


Figura 4.16: Icono y menú del bloque Variable delay

- **Cscope (Figura 4.17):**

Este es el bloque funciona muy parecido al bloque Scope de Simulink. En el menú se puede programar la longitud del eje Y y el periodo de actualización del eje X.

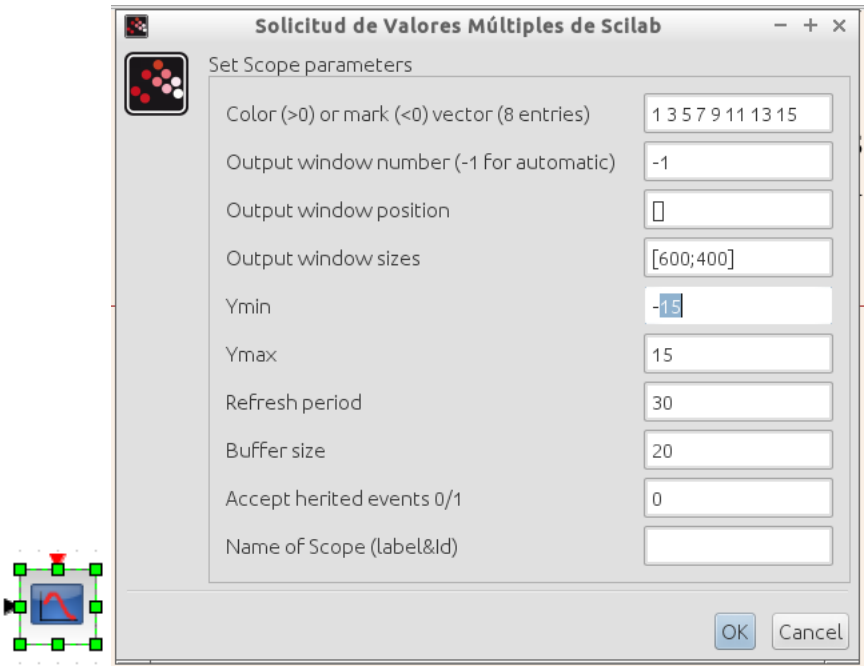


Figura 4.17: Icono y menú del bloque Cscope

- **PID (Figura 4.18):**
  Bloque ya preconfigurado, donde lo único que tendremos que hacer es programar los valores K, 1/Ti y Td en el menú del bloque.

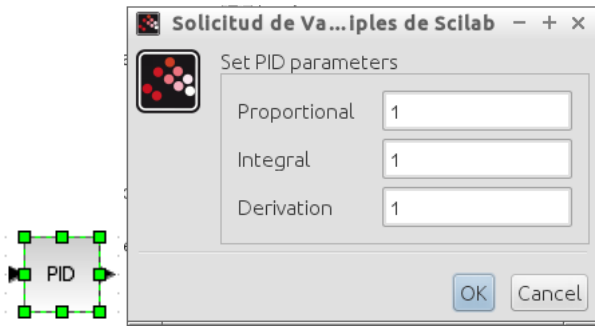


Figura 4.18: Icono y menú del bloque PID

- **Step_Function (Figura 4.19):**
  Bloque utilizado para dar una entrada escalón a nuestro sistema de control. Programaremos los valores inicial, final y el inicio del escalón.

Figura 4.19: Icono y menú Step_Function

- **Scifun_block (Figura 4.20):**
  En este bloque se pueden programar funciones que estén integradas en el lenguaje Scilab. Lo utilizaremos para la comunicación por puerto serie entre Xcos y Arduino. La ausencia de bloque de puerto serie en Xcos nos lleva a utilizar este método para el desarrollo de nuestro sistema de control. A continuación explicaremos la programación realizada en este bloque:

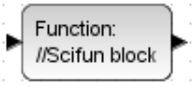


Figura 4.20: Icono Scifun

1. En el menú nº1 (Figura 4.21) del bloque encontramos distintos cuadros donde introducir los valores que necesitamos para programar la naturaleza de nuestro bloque: cantidad de entradas, cantidad de salidas, entradas y salidas de reloj, estados iniciales discretos y continuos, etc. En nuestro bloque utilizaremos dos entradas, dos salidas y una entrada de reloj.

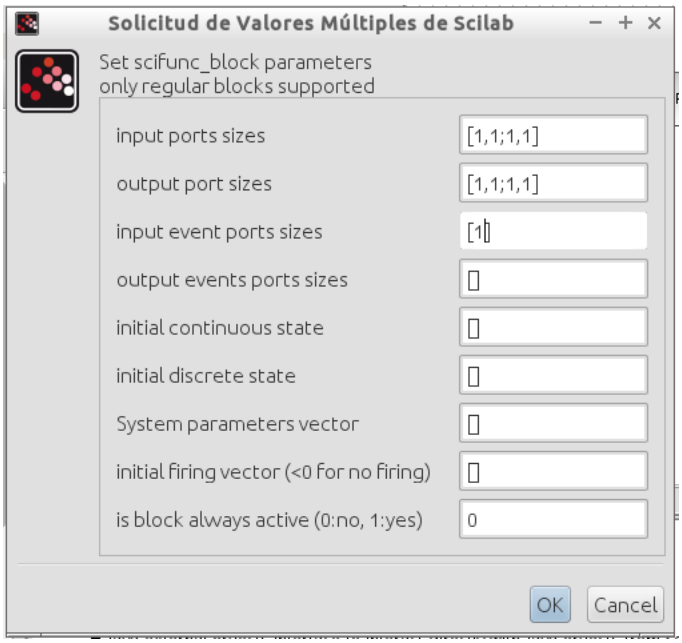


Figura 4.21: Menú nº1 del bloque Scilab

2. En el menú nº2 (Figura 4.22) desarrollaremos el script de comunicación de nuestro sistema. Utilizaremos las funciones de comunicación serie explicadas en la sección de comunicación.

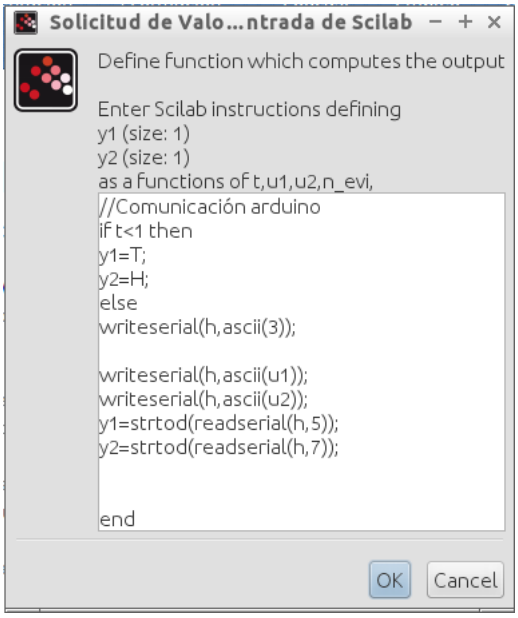


Figura 4.22: Menú nº2 Bloque Scifun

3. El menú nº3 (Figura 4.23) se utiliza para declaración de variables y sentencias antes del inicio

del bloque.

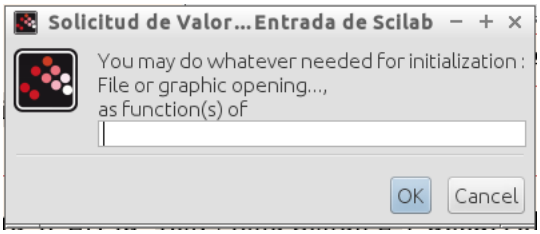


Figura 4.23: Menú Scifun nº3

4. En el menú nº4 (Figura 4.24) se utiliza para declarar sentencias para que se realicen después de la ejecución del bloque.

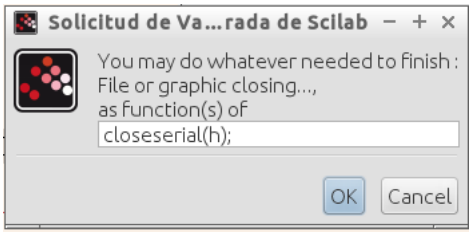


Figura 4.24: Menú Scifun nº4

5. En el menú nº5 (Figura 4.25) fijamos las restricciones de las entradas y de las salidas. En nuestro caso lo tendremos en blanco.

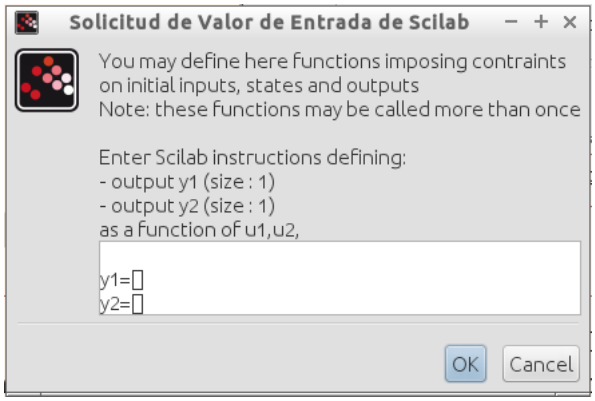


Figura 4.25: Menú Scifun nº5

A continuación mostraremos un ejemplo del sistema de control realizado en Xcos para el control de temperatura de nuestra maqueta. en la siguiente figura (4.26) podemos ver todos los bloques explicados anteriormente.

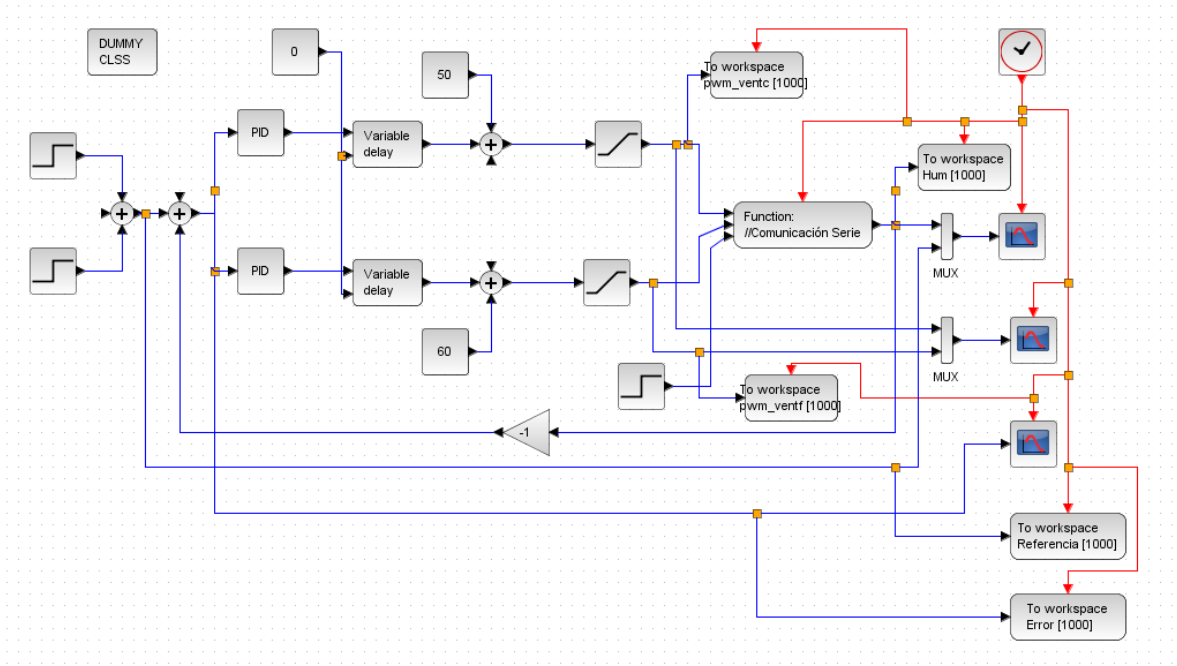


Figura 4.26: Ejemplo Xcos

Capítulo 5

Experimentación.

Este capítulo consistirá en la muestra y explicación de los resultados obtenidos en la etapa de identificación y en la de control.

5.1. Estructura del sistema.

Antes de ir a los resultados, es conveniente explicar como hemos estructurado nuestro sistema de control. Esta elección se ha hecho comprobando previamente la influencia de los distintos actuadores en nuestro sistema (Figura 5.1).

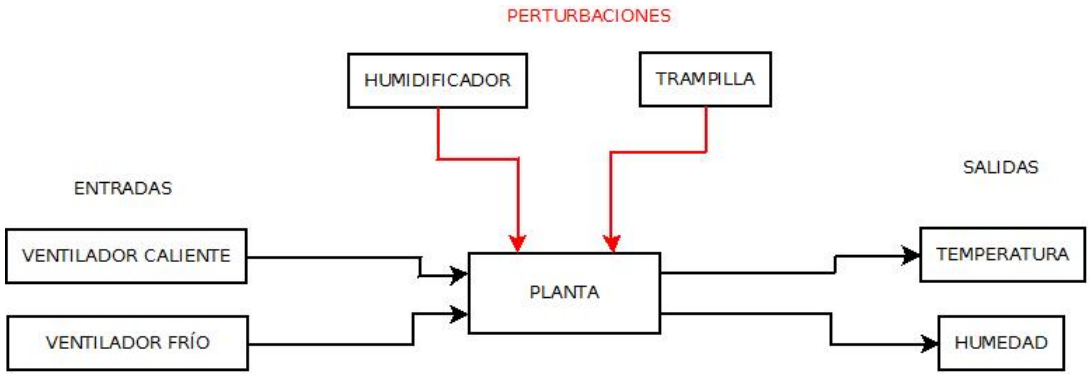


Figura 5.1: Esquema de la estructura del sistema

- **Salidas:**
  Las salidas del sistema o variables a controlar son la temperatura y la humedad provenientes del interior de la maqueta.

- **Entradas:**
  Las entradas del sistema serán la tensión en el ventilador que introducirá aire caliente proveniente de la bombilla y la tensión en el ventilador que introduce aire del exterior. A partir de ahora

designaremos a estos como "ventilador caliente" y "ventilador frío". El ventilador caliente se utilizará para subir la temperatura, mientras que el ventilador frío se utilizará para disminuirla. La bombilla calefactora mantiene un valor constante a la mitad de su potencia. El ventilador caliente se encarga de suministrar mayor o menor flujo de calor. La consideración de la potencia de la bombilla como una tercera entrada hubiera supuesto una dificultad añadida que no modificaría el sistema multivariable que ya tenemos.

- **Perturbaciones:**
  Consideraremos perturbaciones de nuestro sistema el humedecedor y la trampilla automatizada.

5.2. Identificación.

Este proceso consiste en realizar diversos experimentos para identificar el modelo matemático que más concuerda con la dinámica del sistema. Este sistema tiene la singularidad de que es muy lento, por lo tanto, los experimentos a realizar para que se visualicen cambios sustanciales y el sistema llegue a un cierto punto de estabilidad deben de ser largos.

5.2.1. Tiempo de muestreo.

Primero debemos realizar un experimento inicial para saber si nuestro sistema trabaja a tiempo real. Se añadió al software de identificación un código para cronometrar el tiempo de muestreo mediante las funciones tic()-toc() en el bucle de lectura-escritura. Los resultados fueron los siguientes (Figura 5.2):

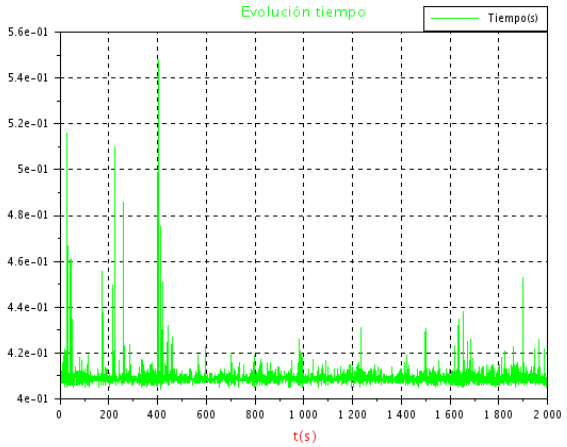


Figura 5.2: Tiempo de muestreo

Aunque aparentemente la gráfica tenga variaciones de tiempo muy altas, lo importante es comprobar que la media de los valores se aproxime al tiempo de muestreo programado en nuestro software. El

comando “mean() ”(Figura 5.3) de Scilab nos dará el resultado de nuestra media.

```
-->mean(time)
ans  =

  0.4093542
```

Figura 5.3: Tiempo de muestreo medio

El resultado es satisfactorio,ya que ronda los 410ms y el tiempo de muestreo programado es de 400ms.

5.2.2. Identificación de la planta.

Para identificar las dinámicas de la planta realizaremos un experimento en el que daremos un escalón a cada una de las entradas (Figura 5.4). Estudiaremos los valores final e inicial, el tiempo de establecimiento y el tiempo de retardo de las variables de salida (temperatura y humedad).

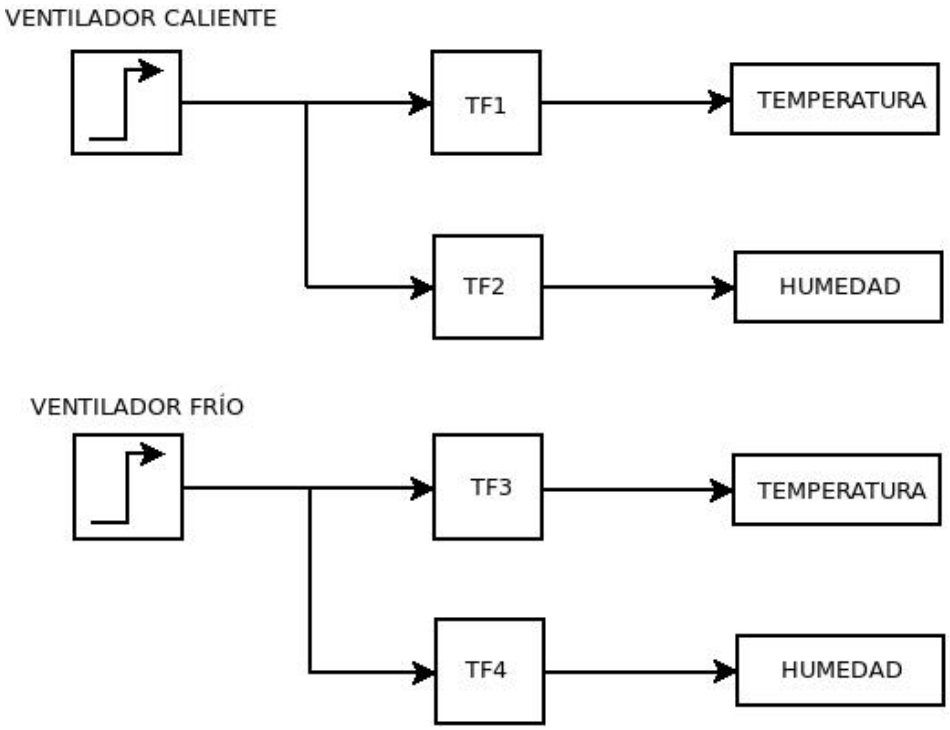
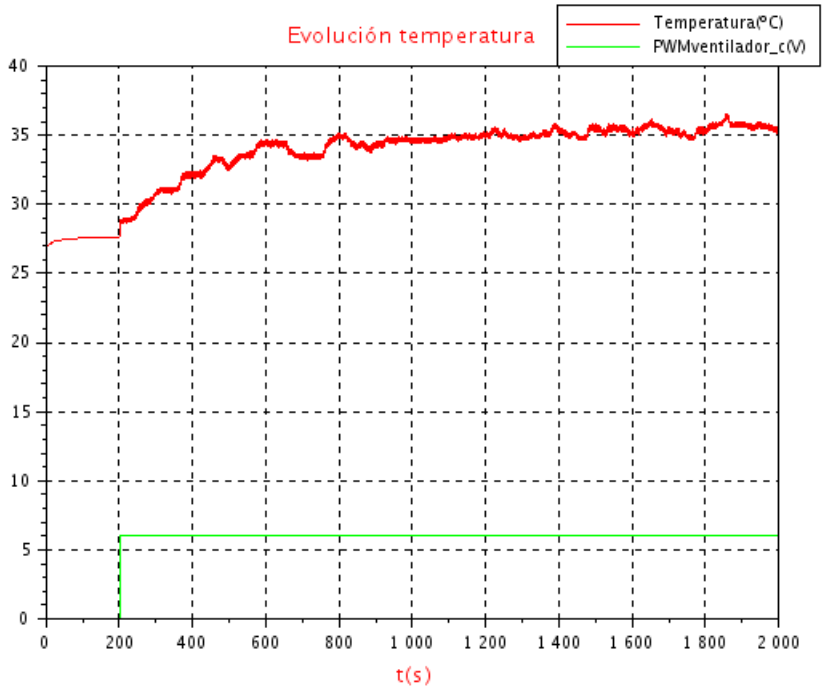
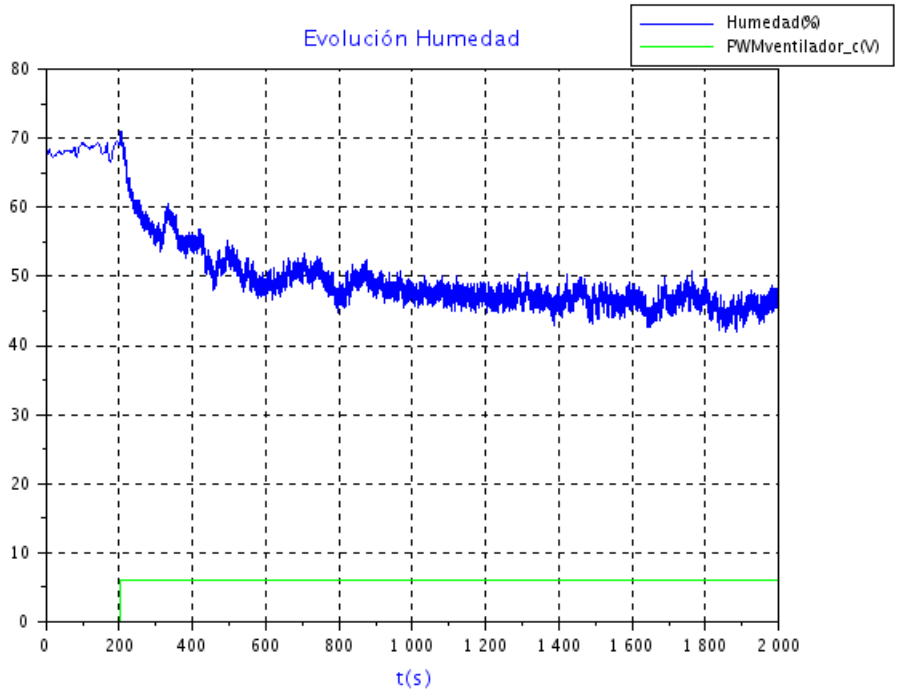


Figura 5.4: Diagrama de bloques de la identificación.

5.2.2.1. Ventilador Aire Caliente .



(a) Gráfica de evolución de la temperatura.



(b) Gráfica de evolución de la humedad.

Figura 5.5: Gráficas identificación Ventilador aire caliente.

Como se puede apreciar en las gráficas(Figura 5.5), la dinámica de las dos variables de salida se pueden identificar como un primer orden con retardo. También se puede ver que la relación entre ambas variables es inversamente proporcional.

Una vez obtenidas estas gráficas procederemos a realizar el cálculo de las funciones de transferencia de cada variable. Los valores utilizados se sacan de la misma gráfica.

Temperatura (Figura 5.6):

$t_{imp} = 200s$;Tiempo de inicio del pulso
$t_0 = 1s$; Retardo
$u(t) = 128$; Valor PWM introducido al ventilador.
$T_0 = 27^{o}\text{C}$
$T = 35,5^{o}\text{C}$
$\triangle T = T - T_0 = 35,5 - 27 = 8,5^{o}\text{C}$
$G = \frac{\triangle T}{U} = \frac{8,5}{128}$
$T(\tau) = 0{,}632 \cdot \triangle T + T_0 = 0{,}632 \cdot 8,5 + 27 = 32,72^{o}\text{C}$
$t = 440$; instante de tiempo donde aparece $T(\tau)$ en la gráfica.
$\tau = t - t_{imp} = 440 - 200 = 240s$

$$\mathbf{u(s) = \frac{8{,}5/128}{240s + 1} e^{-s}}$$

Figura 5.6: Cálculo F.T. Temperatura

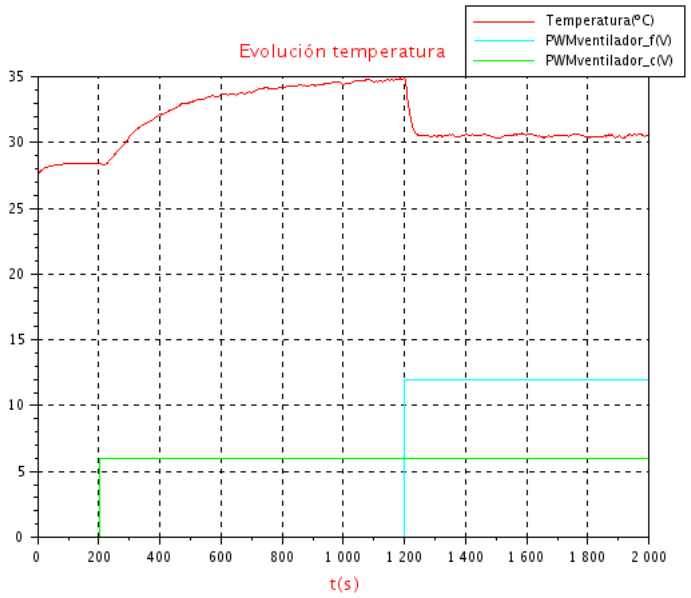
Humedad (Figura 5.7):

$t_{imp} = 200s$;Tiempo de inicio del pulso
$t_0 = 5s$; Retardo
$u(t) = 128$; Valor PWM introducido al ventilador.
$H_0 = 71\,\%$
$H = 46\,\%$
$\triangle H = H - H_0 = 46 - 71 = -15\,\%$
$G = \frac{\triangle H}{U} = -\frac{15}{128}$
$H(\tau) = H_0 - 0{,}632 \cdot \triangle H = 71 - 0{,}632 \cdot 15 = 61{,}52\,\%$
$t = 230$; instante de tiempo donde aparece $H(\tau)$ en la gráfica.
$\tau = t - t_{imp} = 230 - 200 = 30s$

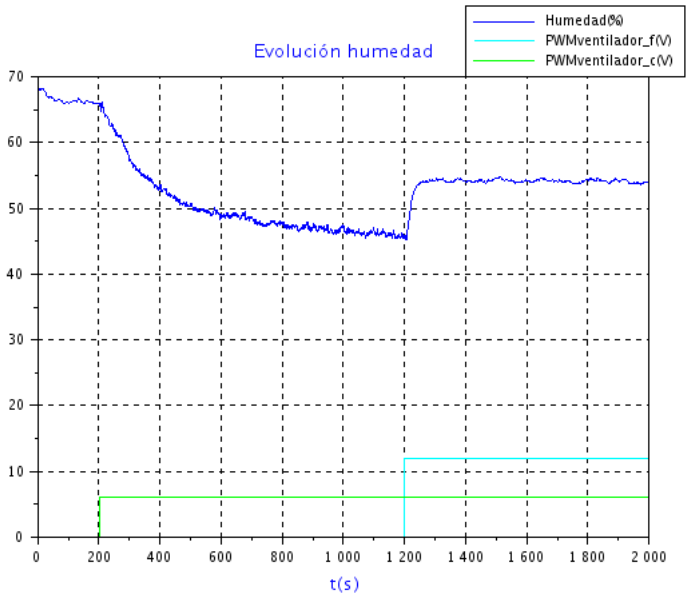
$$\mathbf{u(s) = \frac{-15/128}{30s + 1} e^{-5s}}$$

Figura 5.7: Cálculo F.T. Humedad.

5.2.2.2. Ventilador aire frío.



(a) Gráfica Temperatura



(b) Gráfica Humedad

Figura 5.8: Gráficas identificación ventilador aire frío

En esta identificación hemos cambiado el diseño del experimento. Hemos mantenido constante el ventilador de aire caliente y al cabo de un tiempo para que la señal se estabilice, hemos dado una

entrada escalón al ventilador frío a la máxima potencia(12V). Aquí se puede apreciar el efecto tan rápido y pronunciado que tiene el ventilador de aire frío sobre el sistema (Figura 5.8).

Como en el apartado anterior, procederemos a calcular las funciones de transferencia de cada gráfica:
Temperatura (Figura 5.9):

$t_{imp} = 1200seg$;Tiempo de inicio del pulso
$t_0 = 1s$; Retardo
$u(t) = 255$; Valor PWM introducido al ventilador.
$T_0 = 35^{o}$C
$T = 31^{o}$C
$\triangle T = T - T_0 = 31 - 35 = -4^{o}$C
$G = \frac{\triangle T}{U} = \frac{-4}{255}$
$T(\tau) = 0{,}632 \cdot \triangle T + T_0 = -0{,}632 \cdot 4 + 35 = 32{,}47^{o}$C
$t = 1230$; instante de tiempo donde aparece $T(\tau)$ en la gráfica.
$\tau = t - t_{imp} = 1230 - 1200 = 30s$

$$\mathbf{u(s) = \frac{^{-4}/_{255}}{30s + 1} e^{-s}}$$

Figura 5.9: Cálculo F.T. Temperatura

Humedad (Figura 5.10):

$t_{imp} = 1200s$;Tiempo de inicio del pulso
$t_0 = 5s$; Retardo
$u(t) = 255$; Valor PWM introducido al ventilador.
$H_0 = 45\,\%$
$H = 54\,\%$
$\triangle H = H - H_0 = 54 - 45 = 9\,\%$
$G = \frac{\triangle H}{U} = \frac{9}{255}$
$H(\tau) = H_0 + 0{,}632 \cdot \triangle H = 45 + 0{,}632 \cdot 9 = 50{,}68\,\%$
$t = 1220$; instante de tiempo donde aparece $H(\tau)$ en la gráfica.
$\tau = t - t_{imp} = 1220 - 1200 = 20s$

$$\mathbf{u(s) = \frac{^{9}/_{255}}{20s + 1} e^{-5s}}$$

Figura 5.10: Cálculo F.T. Humedad.

5.3. Control.

En esta sección presentaremos los distintos tipos de control diseñados para la maqueta. Para la sintonización de los PIDs que conforman nuestros sistemas de control hemos utilizado el método empírico, ya que las sintonizaciones por Ziegler-Nichols no nos daban resultados satisfactorios. Se

han ido probando diferentes sintonizaciones hasta dar con una que cumpliera unos requisitos mínimos de control.

En esta etapa ha aflorado otro problema que en un principio era inexistente. La histéresis de los ventiladores. Esta histéresis generaba zonas muertas en nuestro sistema, lo que hacia imposible el control de las variables. Con unos cambios en la estructura del control añadiendo offsets y cambiando uno de los ventiladores se consiguió subsanar este error.

Nuestro sistema de control funcionará de dos maneras diferentes: activará el ventilador de aire caliente cuando necesite aumentar la temperatura o disminuir la humedad; o activará el ventilador de aire frío cuando necesitemos una aumento de humedad y necesitemos disminuir la temperatura. Este funcionamiento tiene sus limitaciones: no podremos aumentar humedad ni disminuir temperatura más allá de las variables del ambiente.

Este funcionamiento también implica tener dos funciones de transferencia diferentes para cada variable, por lo tanto, supondrá un controlador PID anexo a cada función de transferencia.

5.3.1. Control monovariable.

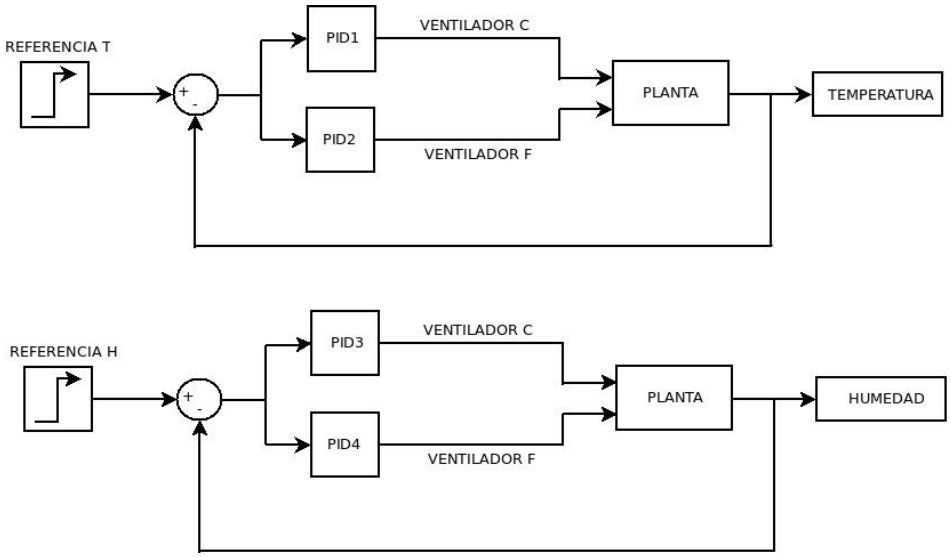


Figura 5.11: Diagrama de bloques del control monovariable.

La estructura de nuestro sistema de control es la siguiente (Figura 5.12):

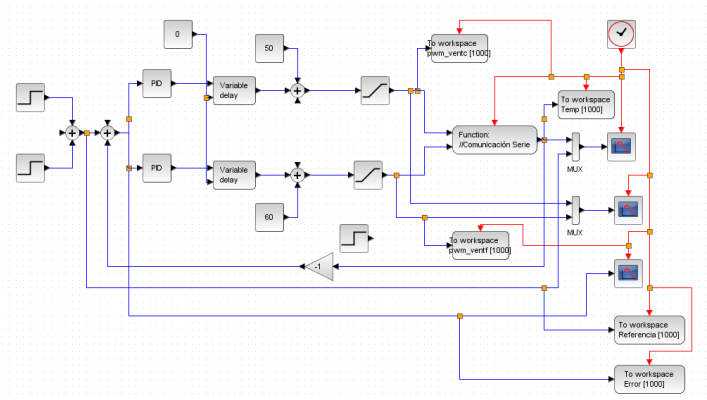


Figura 5.12: Estructura sistema de control de una variable

Como se puede ver en la Figura 5.12, se aprecia lo comentado anteriormente sobre la necesidad de dos controladores PID, uno para cada señal de control. También se ha utilizado un bloque de saturación (Figura 5.13) para evitar que la señal de control pase del rango permitido por el ventilador.

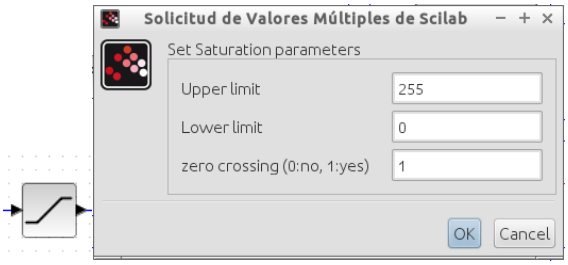


Figura 5.13: Bloque y menú bloque saturación

5.3.1.1. Control temperatura.

Para el control de esta variable nos decantamos por un control PI en cada uno de los ventiladores, la sintonización para cada uno es la misma solo que de signo opuesto (Figura 5.14):

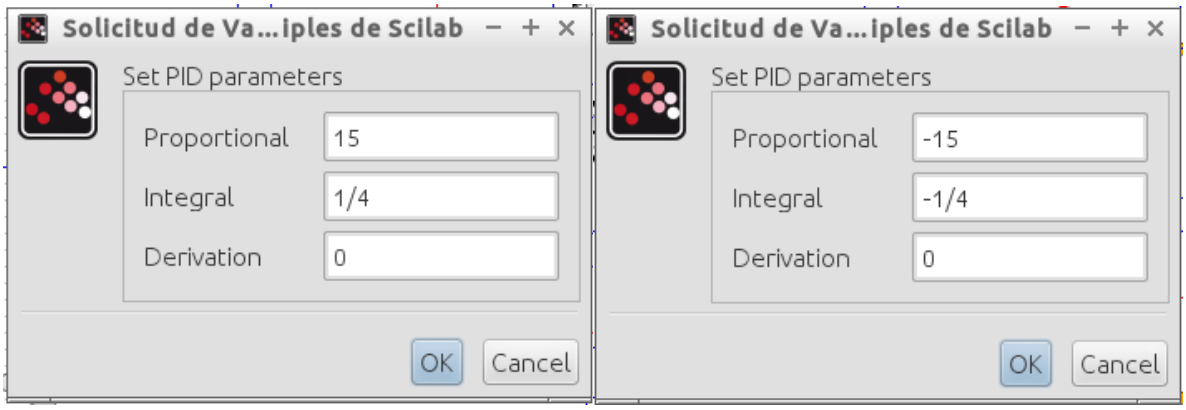


Figura 5.14: Sintonización PID temperatura

Los resultados obtenidos con esta sintonización fueron los siguientes (Figura 5.15):

Como se puede apreciar en la gráfica, con esta sintonización obtenemos un control sin sobrepico y con una buena respuesta al cambio de referencia.

5.3.1.2. Control humedad.

Esta variable ha sido más difícil de controlar, hemos necesitado de un controlador PID y como se podrá comprobar, los resultados no han sido tan limpios que con la temperatura. La sintonización requerida para el control se ilustra en la siguiente Figura (5.16):

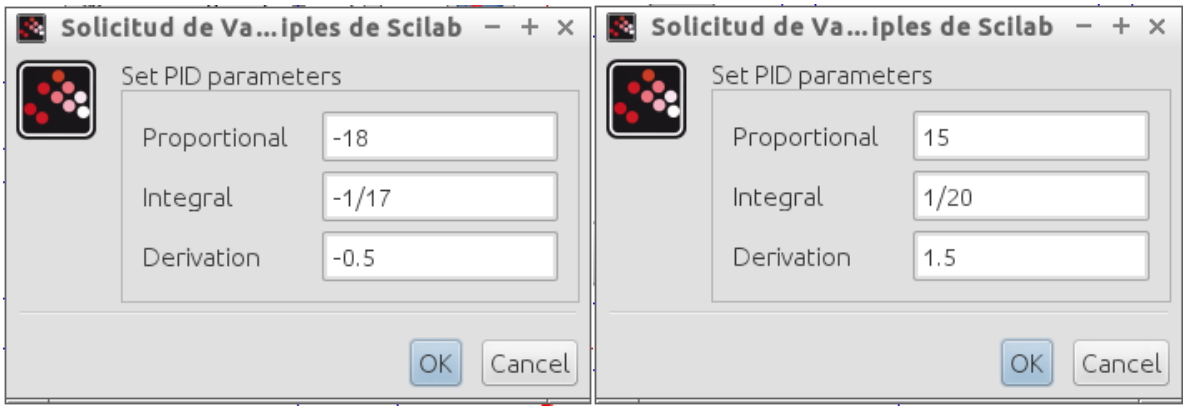
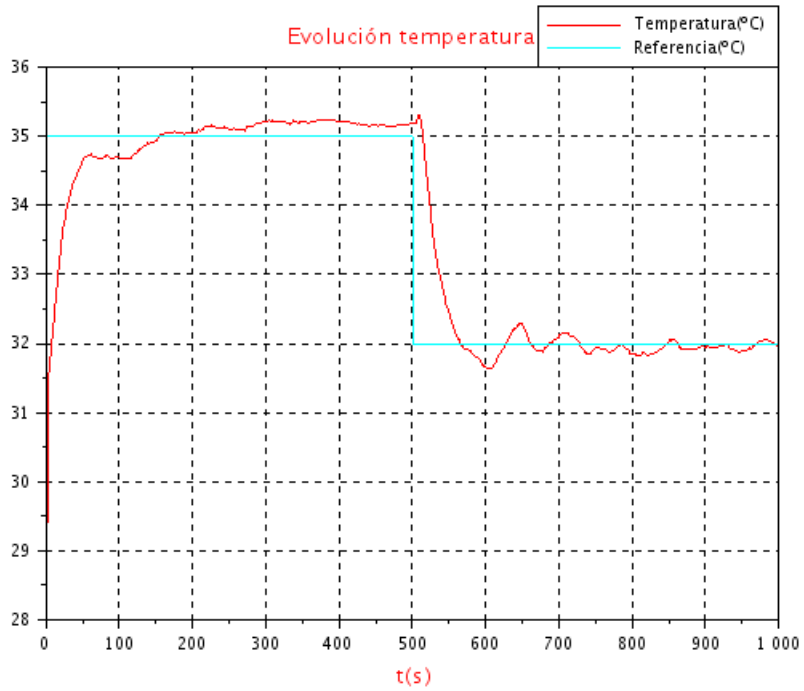


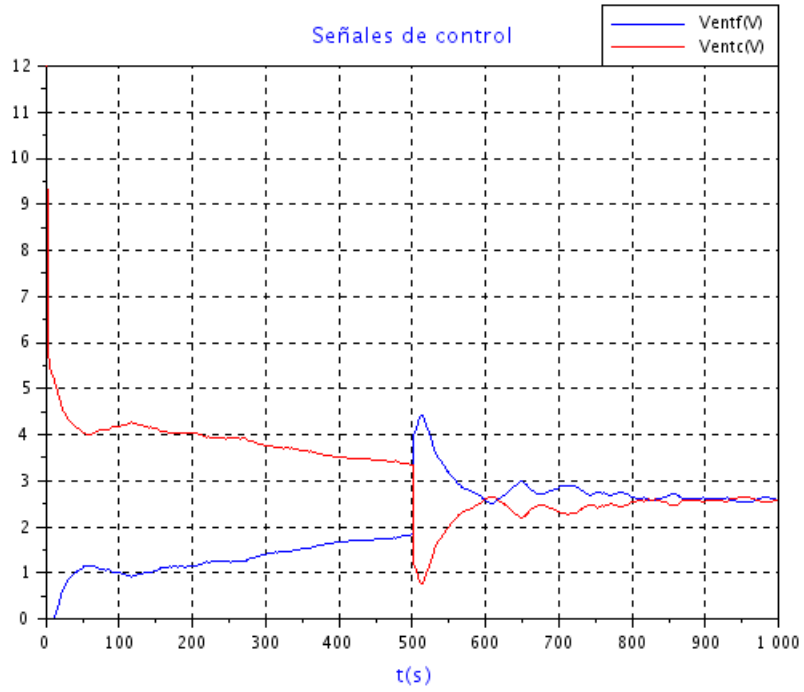
Figura 5.16: Sintonización PID humedad

Los resultados se visualizan a continuación (Figura 5.17):

La gráfica ilustra como tenemos un sobrepico al inicio del control que consigue subsanarse, además la respuesta al cambio en la referencia es buena y el sobrepico desaparece.

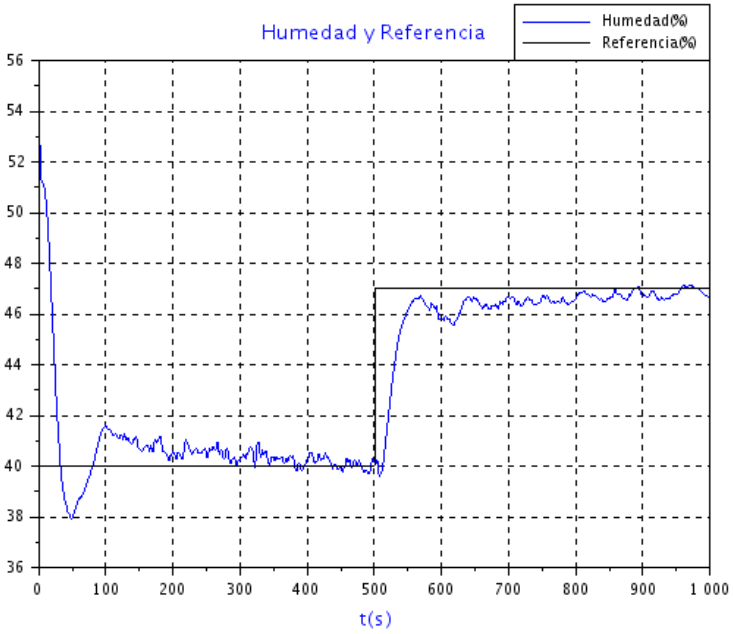


(a) Temperatura y referencia

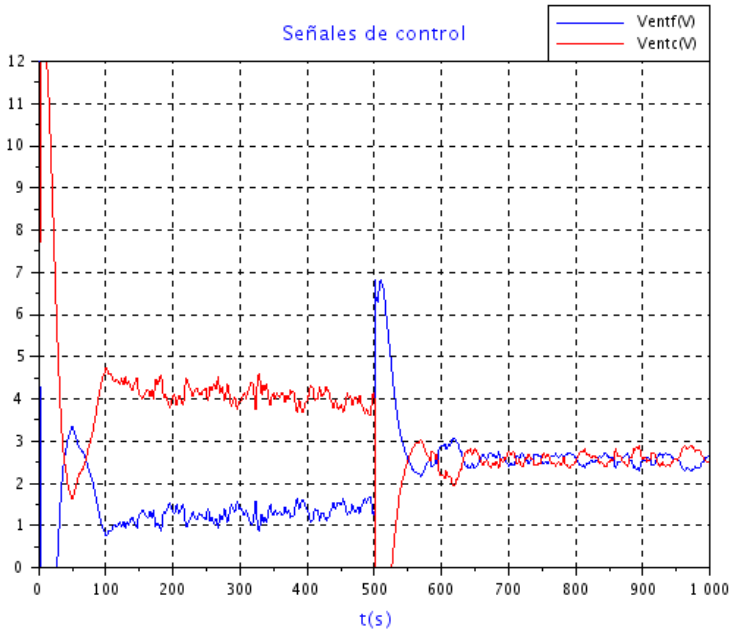


(b) Señales de control

Figura 5.15: Control PI Temperatura



(a) Humedad y referencia



(b) Señales de control

Figura 5.17: Control PID Humedad

5.3.2. Control multivariable.

Como se ha visto en las gráficas de identificación, la temperatura y la humedad tienen una relación inversa entre sí. Cuando intentas incrementar temperatura, disminuyes humedad y viceversa. Por lo que sin un tercer actuador encargado de aumentar humedad, es algo complicado. Mientras alguien lee esto se puede preguntar por qué no utilizamos el humedecedor. En realidad, el humedecedor sólamente sirve para aumentar la humedad a muy largo plazo, lo único a lo que afectaría sería a las condiciones iniciales de nuestro sistema, y poco más. Si mantenemos más húmedo el suelo de la maqueta que el aire de exterior, al poner en funcionamiento el ventilador frío nos disminuiría esa humedad hasta la humedad ambiente, así que no tendría demasiado sentido utilizarlo.

Para intentar realizar el control simultaneo de las dos señales hemos diseñado dos estructuras, una donde combinamos los cuatro controladores que hemos tenido anteriormente, y la otra donde que consiste en un control con desacoplo.

5.3.2.1. Control multivariable sin desacoplo.

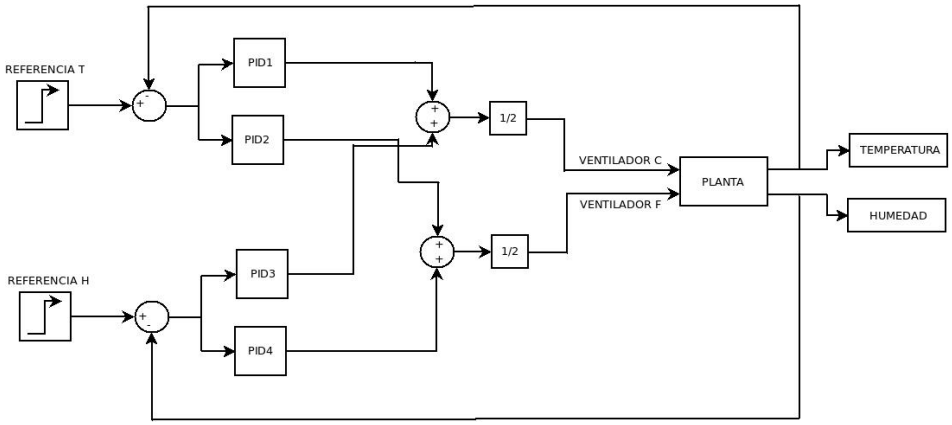


Figura 5.18: Diagrama de bloques del control multivariable sin desacoplo.

Esta estructura se basa en las realizadas para el control monovariable, sumando las señales de control correspondientes a cada ventilador y realizandole la media aritmética. La estructura diseñada en Xcos queda así (Figura 5.19):

A continuación mostraremos los resultados obtenidos con esta estructura (Figuras 5.20 y 5.21)

Como se puede apreciar en las gráficas, el control no es muy preciso. Hemos intentado dividir el experimento en dos partes. Una parte donde la relación temperatura-humedad se corresponde aproximádamente con las obtenidas en los anteriores experimentos. Y otra parte donde intentamos romper esa relación de valores. Entre el 1000s y 1500s es donde intentamos conseguir 30ºC a 36 % de humedad. Como estos valores rompen la relación que hay entre temperatura y humedad, el sistema se comporta de forma errónea.

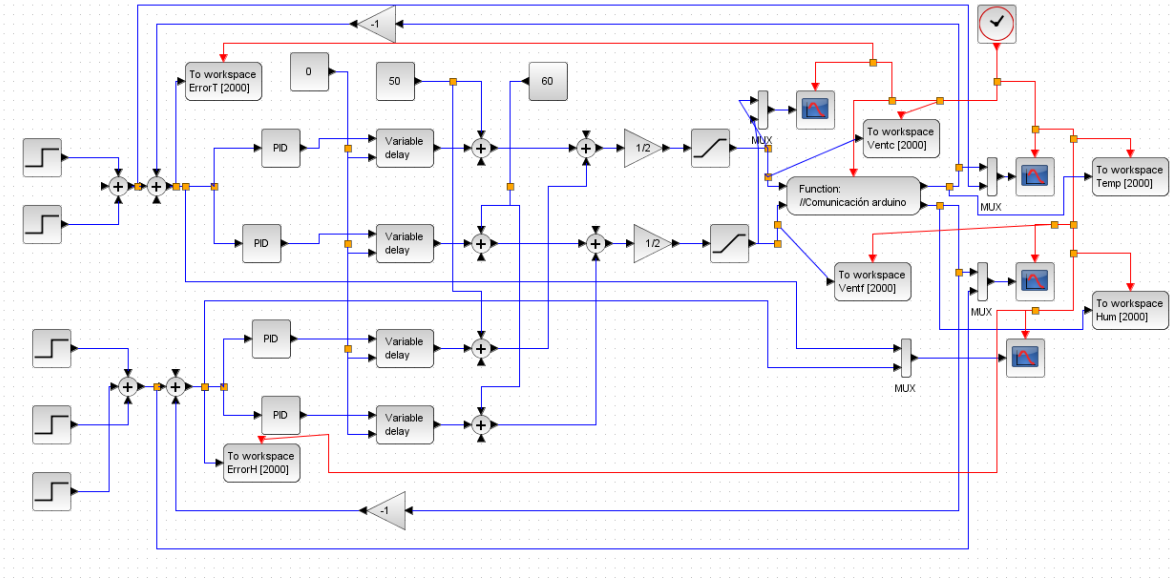


Figura 5.19: Control Temperatura y Humedad

5.3.2.2. Control multivariable con desacoplo.

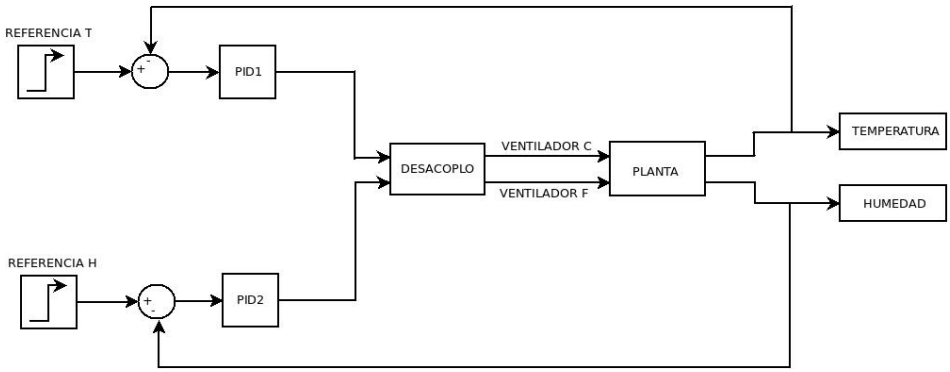
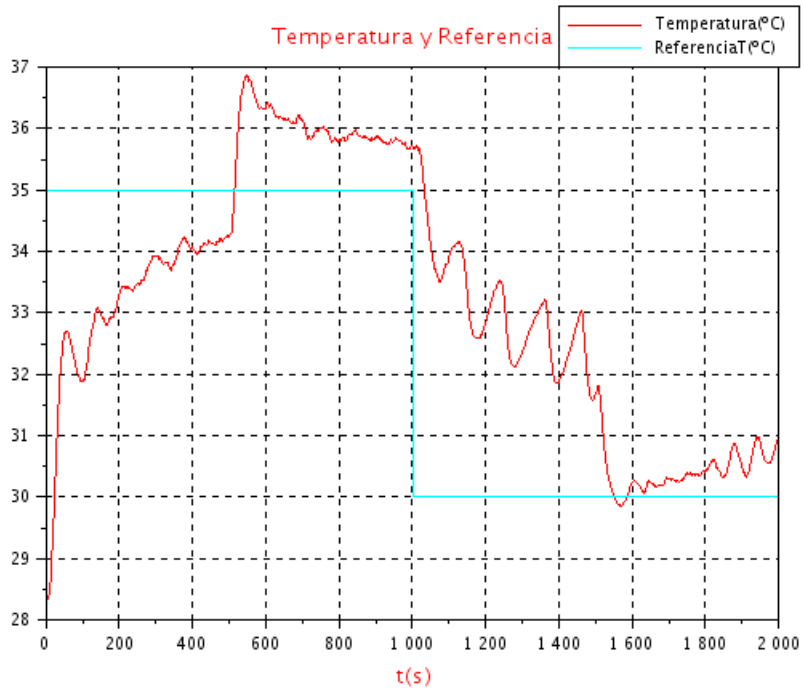
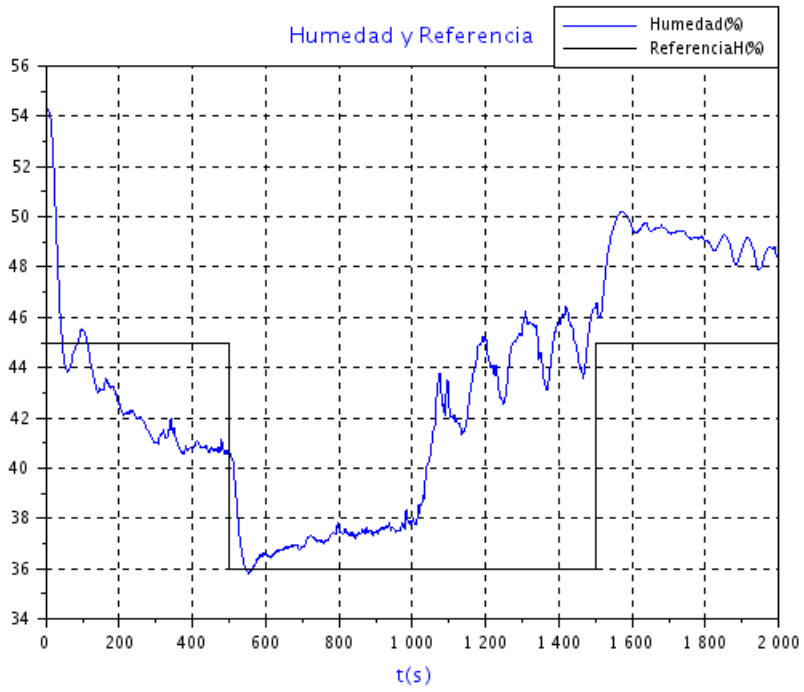


Figura 5.22: Diagrama de bloques del control con desacoplo.

El desacoplo es una técnica utilizada para subsanar los errores generados por las interacciones entre lazos. La idea consiste en diseñar un controlador que reduzca los efectos de la interacción. Esto puede ser llevado a cabo por un compensador que interactúa con las señales enviadas del controlador a la entrada de nuestro proceso. A continuación desarrollaremos el modelo matemático para un sistema con dos entradas y dos salidas, pero también se puede realizar con otros sistemas con la misma cantidad de entradas y salidas.

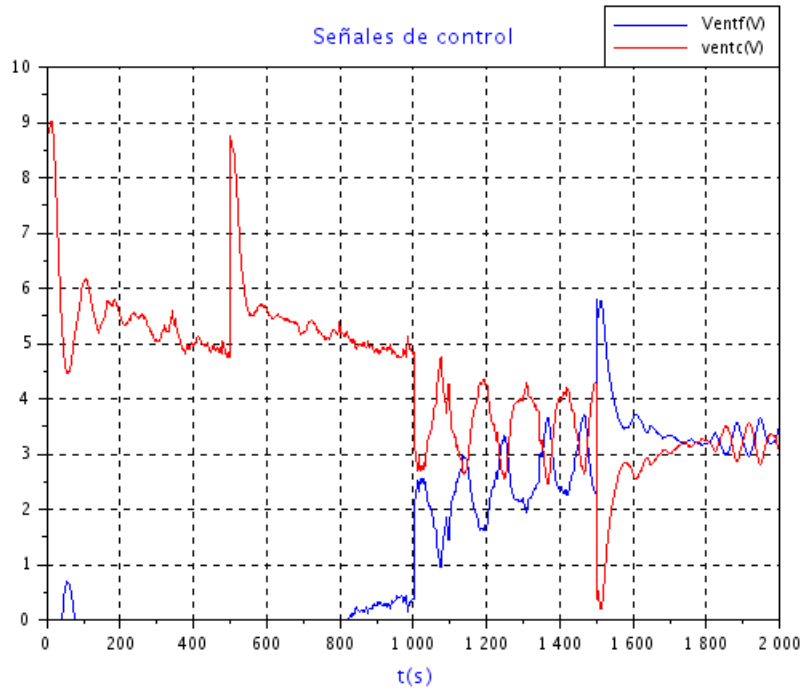


(a) Temperatura y Referencia.

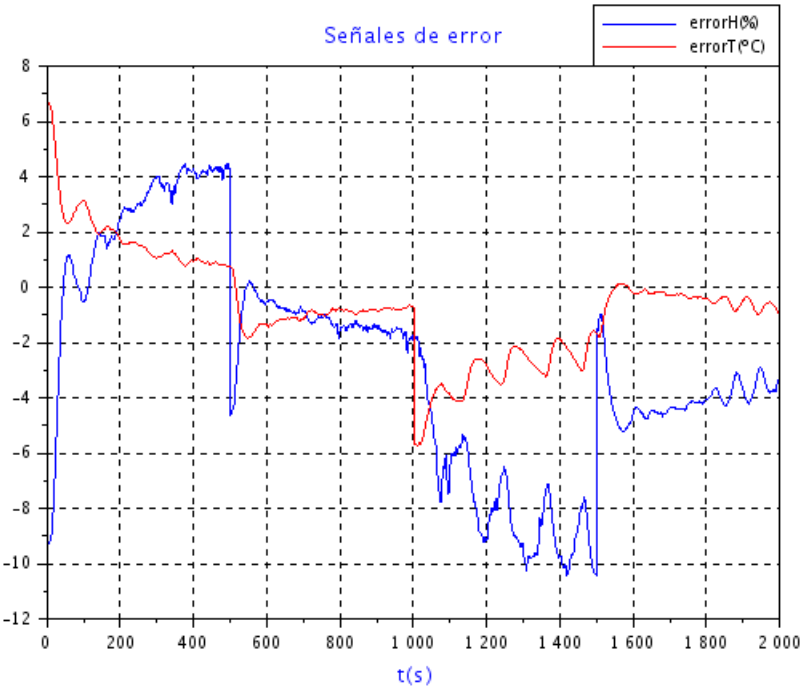


(b) Humedad y Referencia.

Figura 5.20: Control Temperatura y Humedad.



(a) Señales de control



(b) Señales de error

Figura 5.21: Señales de control y error

Sea la matriz del sistema $G(s)$:

$$G = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$$

Tal que:

$$Y = G \cdot U = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

La matriz P es:

$$P = lim_{s \to 0} G(s) = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix}$$

Siendo Y la matriz de variables de salida, P la variable de ganancias del sistema y U la matriz de entradas.
En desacoplo consiste en encontrar una matriz D tal que:

$$D = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} = P^{-1} = \frac{1}{\det P} \begin{pmatrix} p_{22} & -p_{12} \\ -p_{21} & p_{11} \end{pmatrix}$$

Una vez encontrada nuestra matriz D, la nueva señal de control es:

$$U^{'} = D \cdot U = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_1^{'} \\ u_2^{'} \end{pmatrix} = \begin{pmatrix} d_{11} \cdot u_1 + d_{12} \cdot u_2 \\ d_{21} \cdot u_1 + d_{22} \cdot u_2 \end{pmatrix}$$

En nuestro sistema, la matriz P será la matriz de ganancias del sistema:

$$P = \begin{pmatrix} 8{,}5 & 4 \\ 15 & 9 \end{pmatrix}$$

En la primera columna colocamos las ganancias pertenecientes al ventilador caliente y en la segunda las del ventilador frío. A partir de esta matriz de ganancias obtenemos la matriz de desacoplo:

$$D = P^{-1} = \begin{pmatrix} 0{,}545454 & -0{,}242424 \\ -0{,}909090 & 0{,}51515151 \end{pmatrix}$$

Una vez calculada la matriz de desacoplo la introduciremos en nuestra estructura de control en Xcos mediante el bloque Scifun mencionado anteriormente. El código introducido en la función será el siguiente (Figura 5.23):

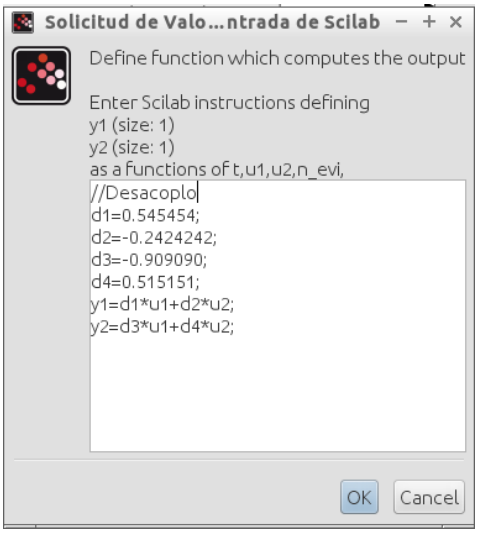


Figura 5.23: Función desacoplo

Para la construcción de la estructura de control del sistema desacoplado elegimos los controladores PID con ganancia positiva y le conectamos el bloque de desacoplo, como aparece a continuación (Figura 5.24):

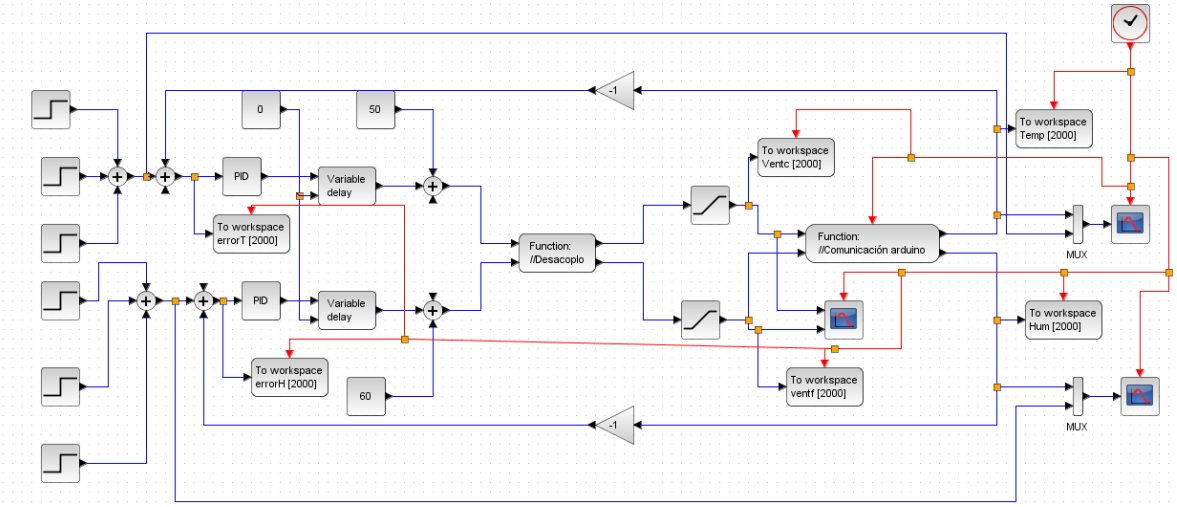
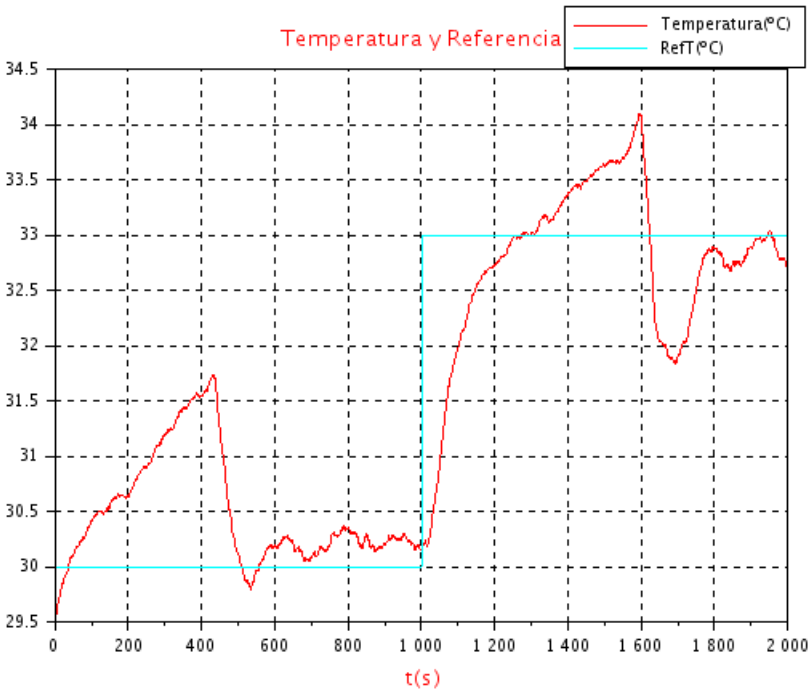


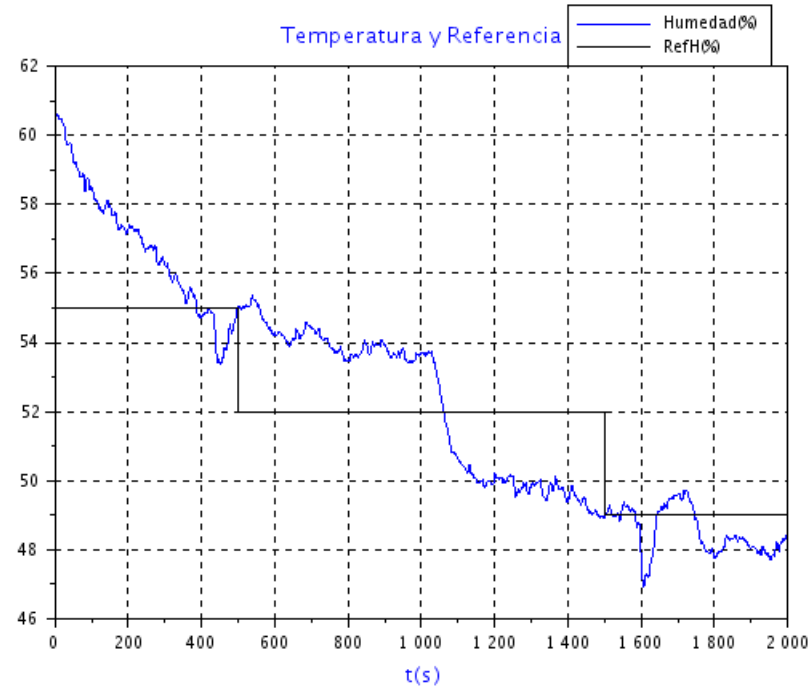
Figura 5.24: Diagrama del sistema con desacoplo

A continuación mostraremos las gráficas resultantes de aplicar esta técnica al control multivariable de la maqueta (Figura 5.25, Figura 5.26):

Como ilustran las gráficas anteriores, hemos mejorado el control en comparación con la estructura anterior. Entre los 500s y los 1000s se puede apreciar como se consigue un control de la temperatura y la humedad intenta disminuir hasta la referencia. En los casos donde la relación de las variables no es

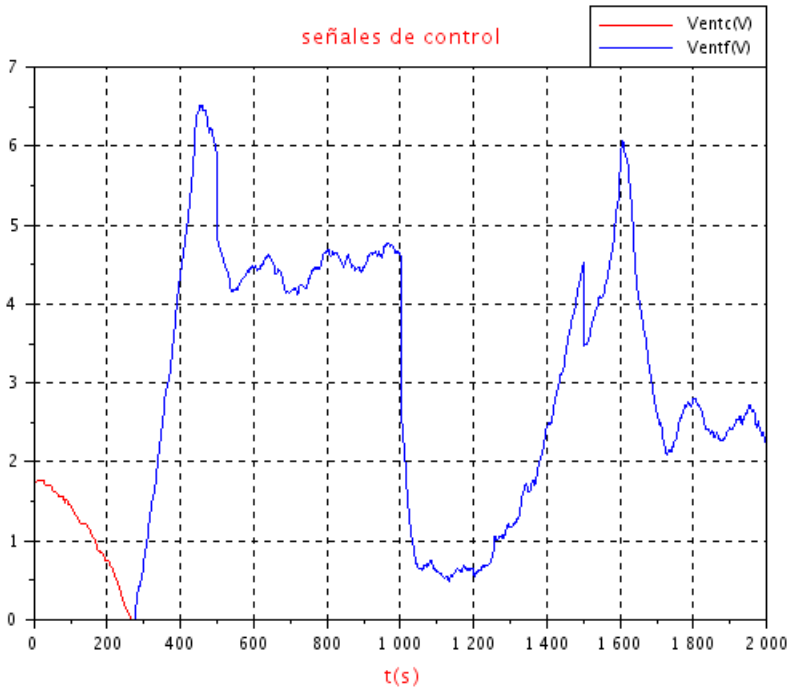


(a) Temperatura y Referencia

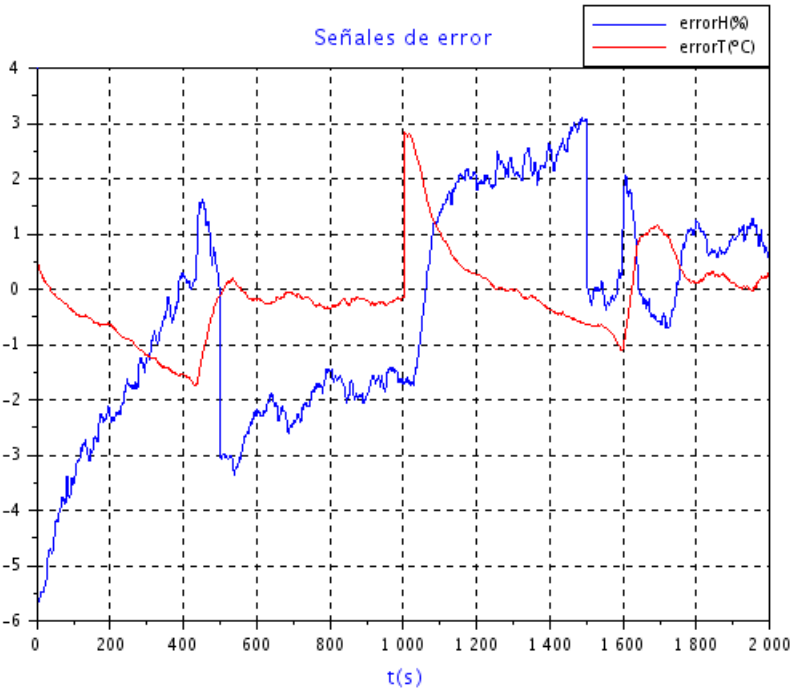


(b) Humedad y Referencia

Figura 5.25: Temperatura y Humedad con desacoplo



(a) Señales de control



(b) Señales de error

Figura 5.26: Señales de control y error

favorable con respecto a la referencia, como la fase de los 1500s a los 2000s se aprecia como el sistema hace sobrepicos, pero tanto temperatura como humedad intentan estabilizarse en una unidad o dos por debajo de la referencia marcada.

Capítulo 6

Conclusiones y futuros trabajos.

6.1. Conclusiones

En este apartado analizaremos si hemos cumplido los distintos objetivos que se proponían para nuestro proyecto:

- **Realización de todo el proyecto con Open Source y estudio de un abanico de alternativas de este tipo en el ámbito de la ingeniería:**
  Hemos dedicado un capítulo completo a la explicación sobre qué es el Open Source y hemos mencionado muchos de los programas más utilizados. Hemos descrito también distintos proyectos de Open Hardware y cómo las empresas de software propietario han introducido el modelo comunitario en su forma de trabajo.

- **Desarrollar y completar la maqueta de invernadero qeu actuará como planta:**
  Hemos explicado con detalle la composición de nuestra maqueta y los cambios realizados en ella. También hemos explicado la electrónica necesaria para el manejo de la maqueta y el paso a paso para obtener la trarjeta PCB.

- **Desarrollo del software necesario para nuestro proyecto:**
  Se ha explicado todo el código desarrollado relacionado con la comunicación, la identificación y el control de nuestro sistema.

- **Experimentación y Control:**
  Aquí hemos mostrado los resultados obtenidos en los distintos sistemas de control diseñados. Se ha diseñado satisfactoriamente el control monovariable de temperatura y humedad por separado. Además del control multivariable sin desacoplo y desacoplo.

A modo de reflexión personal, una vez finalizado el proyecto toca hablar sobre lo que hemos realizado. Desde el punto de vista de los objetivos hemos cumplido las expectativas a pesar de tener algunos problemas. Y es que sobre el papel es sencillo trabajar, pero para trabajar sobre el terreno hace falta

mancharse las manos. Gran cantidad de errores humanos, fallos estúpidos de novato, dolores de cabeza... Aunque al final todo merece la pena cuando ves el objetivo cumplido.

En cuanto al desarrollo del software, todo lo que tenía que ver con Arduino ha sido bastante sencillo. En la red hay gran cantidad de información útil en la que poder consultar todas las dudas necesarias para resolver muchos problemas. No así para Scilab, donde el desarrollo todavía es algo pobre si lo comparamos con Matlab y Labview, lo que hace que su uso sea residual y no haya información suficiente y de calidad. Aun así este escollo también ha sido superado y hemos conseguido un desarrollo de software a la medida de nuestras exigencias.

La parte más divertida sin duda ha sido el descubrimiento de la tecnología Open Source. La idea de realizar todo el proyecto en Open Source fue casi de casualidad y es sin duda lo que más ha enriquecido la realización del proyecto. Gran parte del software que he aprendido a utilizar para este proyecto lo utilizo continuamente en mi vida cotidiana. Y me ha aportado una visión de la tecnología totalmente diferente a la que tenia anteriormente. El Open Source es colaboración y a la vez competitividad, apoyo mutuo y también libertad. Este tipo de tecnología ha hecho que algo que parecía tan complejo como una adquisición de datos se pueda realizar de forma casera y a un coste irrisorio si lo comparámos con los que costaba hace años.

Pero teniendo los pies en la tierra, esta tecnología aun no puede competir con el software propietario. Ni Scilab, ni Libreoffice ni casi ningún software Open Source o libre puede competir con el software propietario más potente de su misma rama. Lo que sí está claro es que todos no necesitamos el software más profesional ni el más puntero para realizar todas las tareas. Un usuario estándar de PC que necesite una suite ofimática y navegar por Internet está más que servido con casi cualquier distribución Linux del mercado. Pero cualquier ingeniero que utilice Autocad no puede utilizar este tipo de distribuciones ya que todavía no hay ningún software que trabaje bien con el .dwg. Así que en la combinación del software propietario y el Open Source se encuentra el equilibrio perfecto para conseguir una gran productividad a bajo coste a la hora de realizar tus proyectos.

6.2. Futuros trabajos y mejoras.

A continuación presentaremos unas cuántas mejoras que se pueden hacer en el futuro y mejorar sustanciálmente la maqueta.

- **Control PID programado en Arduino:**
  Al programar el control PID con el Arduino eliminaríamos al PC como intermediario. Lo que permitiría trabajar en el PC con Scilab mientras el sistema trabaja de forma autónoma. Para la recolección de los datos hay módulos de micro sd para Arduino donde poder almacenar los valores de las variables para después cargarlos en Scilab o Matlab y realizar identificaciones,

gráficas, etc.

- **Instalación de una LCD:**
  Es algo que se barajó para instalar en este proyecto, pero las LCD estándar necesitan muchos pines para su funcionamiento. Hay pantallas LCD que funcionan sólo con cuatro pines. Este elemento nos mostraría los datos de la temperatura y la humedad en tiempo real sin necesidad de ser monitorizado vía PC. También se podría intentar programar la referencia o la sintonización del PID desde la LCD.

- **Estudio de las perturbaciones:**
  Estudio del dimensionamiento del humedecedor y estudio de las condiciones de uso de la perturbación de la trampilla para afrontar nuevos desafios en el control del invernadero.

- **Comunicación Arduino-PC:**
  Para tener más comodidad y eliminar cables, sería una gran opción cambiar la comunicación entre el Arduino y el PC por algún tipo de conexión inalámbrica, vía Wifi o Bluetooth.

Bibliografía

[1] -Katshuiko Ogata: Ingeniería de Control Moderna. 4º Edición. Pearson.

[2] -Stephen L. Cambell, Jean Phillipe Chancelier y Ramire Nikoukhan: Modeling and Simulation in Scilab/Scicos. 1ºEdición .Springer.

[3] -Karl J. Aström , Tore Hägglund: Advanced PID Control. 1º Edición. ISA.

[4] - www.arduino.cc

[5] -www.openeering.com

Apéndice A

Script del GUI de Scilab

```
// This GUI file is generated by guibuilder version 2.2
////////// f=figure('figure_position',[-4,-4],'figure_size',[1288,764],
'auto_resize','on','background',[33],'figure_name',
'Ventana de gráfico número %d'); //////////
delmenu(f.figure_id,gettext('File'))
delmenu(f.figure_id,gettext('?'))
delmenu(f.figure_id,gettext('Tools'))
toolbar(f.figure_id,'off')
handles.dummy=0; handles.frame1=uicontrol(f,'unit','normalized','BackgroundColor',
[0.8,0.8,0.8],'Enable','on','FontAngle','normal','FontName',
'helvetica','FontSize',[12],
'FontUnits','points','FontWeight','normal','ForegroundColor',[0,0,0],
'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.021875,0.7002967,0.228125,0.2997033],'Relief','ridge',
'SliderStep',[0.01,0.1],'String','UnName1','Style','frame','Value',[0],
'VerticalAlignment','middle','Visible','on','Tag','frame1','Callback','')
handles.frame2=uicontrol(f,'unit','normalized','BackgroundColor',[0.8,0.8,0.8],
'Enable','on','FontAngle','normal','FontName','helvetica','FontSize',[12],
'FontUnits','points','FontWeight','normal','ForegroundColor',[0,0,0],
'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.021875,0.0113353,0.228125,0.6889614],'Relief','ridge',
'SliderStep',[0.01,0.1],'String','UnName2','Style','frame','Value',[0],
'VerticalAlignment','middle','Visible','on','Tag','frame2','Callback','')

handles.axe1= newaxes();

handles.axe1.margins = [ 0 0 0 0];
```

```
handles.axe1.axes_bounds = [0.4992188,0.0979228,0.2307812,0.6513353];

handles.axe2= newaxes();

handles.axe2.margins = [ 0 0 0 0];

handles.axe2.axes_bounds=[0.7592188,0.0994065,0.2207813,0.6498516];
handles.ps_open=uicontrol(f,'unit','normalized','BackgroundColor',
[0.6,0.6,0.6],'Enable','on','FontAngle','normal','FontName',
'helvetica','FontSize',[12],
'FontUnits','points','FontWeight','normal','ForegroundColor',[0,0,0],
'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.0554688,0.8531157,0.1515625,0.1008902],'Relief','raised',
'SliderStep',[0.01,0.1],'String','Abrir Puerto Serie','Style','pushbutton',
'Value',[0],'VerticalAlignment','middle','Visible','on','Tag','ps_open',
'Callback','ps_open_callback(handles)')

handles.ps_close=uicontrol(f,'unit','normalized','BackgroundColor',
[0.6,0.6,0.6],'Enable','on','FontAngle','normal','FontName',
'helvetica','FontSize',[12],'FontUnits','points','FontWeight',
'normal','ForegroundColor',[0,0,0],
'HorizontalAlignment','center','ListboxTop',[],
'Max',[1],'Min',[0],'Position',
[0.05625,0.7551929,0.1507812,0.0682493],
'Relief','raised','SliderStep',[0.01,0.1],'String',
'Cerrar puerto serie','Style','pushbutton',
'Value',[0],'VerticalAlignment','middle',
'Visible','on','Tag','ps_close',
'Callback','ps_close_callback(handles)')

handles.edit_periodo=uicontrol(f,'unit','normalized','BackgroundColor',
[0.8,0.8,0.8],'Enable','on','FontAngle','normal','FontName','helvetica',
'FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',
[0,0,0],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.115039,0.6175964,0.1003125,0.0459941],'Relief','sunken',
'SliderStep',[0.01,0.1],'String','1000','Style','edit','Value',[0],
'VerticalAlignment','middle','Visible','on','Tag','edit_periodo','Callback','')

handles.texto_muestreo=uicontrol(f,'unit','normalized','BackgroundColor'
```

```
,[0.8,0.8,0.8],'Enable','on','FontAngle','normal','FontName','helvetica',
'FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',
[0,0,0],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.0289063,0.6246291,0.06875,0.0356083],'Relief','flat',
'SliderStep',[0.01,0.1],'String','Periodo(ms)','Style','text','Value',[0],
'VerticalAlignment','middle','Visible','on','Tag','texto_muestreo','Callback','')

handles.ps_experimento=uicontrol(f,'unit','normalized','BackgroundColor',
[0.6,0.6,0.6],'Enable','on','FontAngle','normal','FontName','helvetica',
'FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',
[0,0,0],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.0545313,0.4456083,0.1623438,0.0696736],'Relief','raised',
'SliderStep',[0.01,0.1],'String','Experimento','Style','pushbutton',
'Value',[0],'VerticalAlignment','middle','Visible','on','Tag',
'ps_experimento','Callback','ps_experimento_callback(handles)')

handles.editar_muestras=uicontrol(f,'unit','normalized','BackgroundColor',
[0.8,0.8,0.8],'Enable','on','FontAngle','normal','FontName','helvetica',
'FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',
[0,0,0],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],
'Position',[0.115039,0.5567656,0.1003125,0.0459941],'Relief','sunken',
'SliderStep',[0.01,0.1],'String','5000','Style','edit','Value',[0],
'VerticalAlignment','middle','Visible','on','Tag','editar_muestras','Callback','')

handles.texto_muestras=uicontrol(f,'unit','normalized','BackgroundColor',
[0.8,0.8,0.8],'Enable','on','FontAngle','normal','FontName',
'helvetica','FontSize',[12],'FontUnits','points','FontWeight',
'normal','ForegroundColor',[0,0,0],'HorizontalAlignment','center',
'ListboxTop',[],'Max',[1],'Min',[0],'Position',
[0.0328125,0.5563798,0.0695313,0.0400593],
'Relief','flat','SliderStep',[0.01,0.1],'String',
'Nº muestras','Style','text','Value',[0],'VerticalAlignment','middle',
'Visible','on','Tag','texto_muestras','Callback','')

handles.axe3= newaxes();
handles.axe3.margins = [ 0 0 0 0];
handles.axe3.axes_bounds = [0.263125,0.1008902,0.2046875,0.6483680];

////////// // Callbacks are defined as below. Please do not delete the
comments as it will be used in coming version //////////
```

```
global h;

function ps_open_callback(handles)
//Write your callback for  ps_open  here
if getos() == "Windows" then
h=openserial('COM5',"9600,n,8,1");
disp("OS Windows reconocido, conexión establecida. ");
elseif getos()=="Linux" then
h=openserial("/dev/ttyACM0","9600,n,8,1");
disp("OS Linux reconocido, conexión establecida. ");
else     disp("Sistema operativo no reconocido.");
end
endfunction
function ps_close_callback(handles)
//Write your callback for  ps_close  here
closeserial(h);
disp("Cierre conexión serie. ");
endfunction
function ps_experimento_callback(handles)
//Write your callback for  ps_experimento  here
t = evstr(get(handles.edit_periodo,'String'));
n = evstr(get(handles.editar_muestras,'String'));
i=1;
pwm_ventf=zeros(1:n);
pwm_ventc=zeros(1:n);
pwm_hum=zeros(1:n);
pwm_bomb=zeros(1:n);
servo1=zeros(1:n);
servo2=zeros(1:n);
temp=zeros(1:n);
hum=zeros(1:n);
time=zeros(1:n);
x=1:n;
for v=1:n
if v<=floor(n/20) then
pwm_ventf(v)=0;
pwm_ventc(v)=0;
pwm_hum(v)=0;
pwm_bomb(v)=255;
```

```
elseif v>floor(n/20) & v<=floor(2*n/10) then
pwm_ventf(v)=0;
pwm_ventc(v)=128;
pwm_hum(v)=0;
pwm_bomb(v)=128;
elseif v>floor(2*n/10) & v<=floor(4*n/10) then
pwm_ventf(v)=0;
pwm_ventc(v)=128;
pwm_hum(v)=0;
pwm_bomb(v)=128;
elseif v>floor(4*n/10) & v<=floor(6*n/10) then
pwm_ventf(v)=0;
pwm_ventc(v)=128;
pwm_hum(v)=0;
pwm_bomb(v)=128;
elseif v>floor(6*n/10) & v<=floor(8*n/10) then
pwm_ventf(v)=0;
pwm_ventc(v)=255;
pwm_hum(v)=0;
pwm_bomb(v)=128;
else
pwm_ventf(v)=0;
pwm_ventc(v)=255;
pwm_hum(v)=0;
pwm_bomb(v)=128;
end
end
while i<=n
tic();
writeserial(h,ascii(1));
temp(i)=strtod(readserial(h,5));
hum(i)=strtod(readserial(h));
writeserial(h,ascii(pwm_ventf(i)));
writeserial(h,ascii(pwm_ventc(i)));
writeserial(h,ascii(pwm_hum(i)));
writeserial(h,ascii(pwm_bomb(i)));
sleep(t);
time(i)=toc();
i=i+1;
end
```

```
plot(handles.axe1, x, temp,'cya+');
plot(handles.axe2, x, hum,'r+');
plot(handles.axe3,x, time,'g--');
save('salidasventc4.sod',temp,hum,time);
save('entradasventc4.sod',n,pwm_ventf,pwm_ventc,pwm_hum,pwm_bomb);
endfunction
```

Apéndice B

Protección de la tarjeta de adquisición.

Para proteger la tarjeta de adquisición hemos utilizado una caja de registro de plástico. Realizando unos cuantos agujeros y limando un poco se ha conseguido acceder a los enchufes y a los conectores a la perfección y a la vez proteger nuestra tarjeta. En las siguiente figuras(B.1, B.2,B.3 y B.4) se pueden ver los resultados.

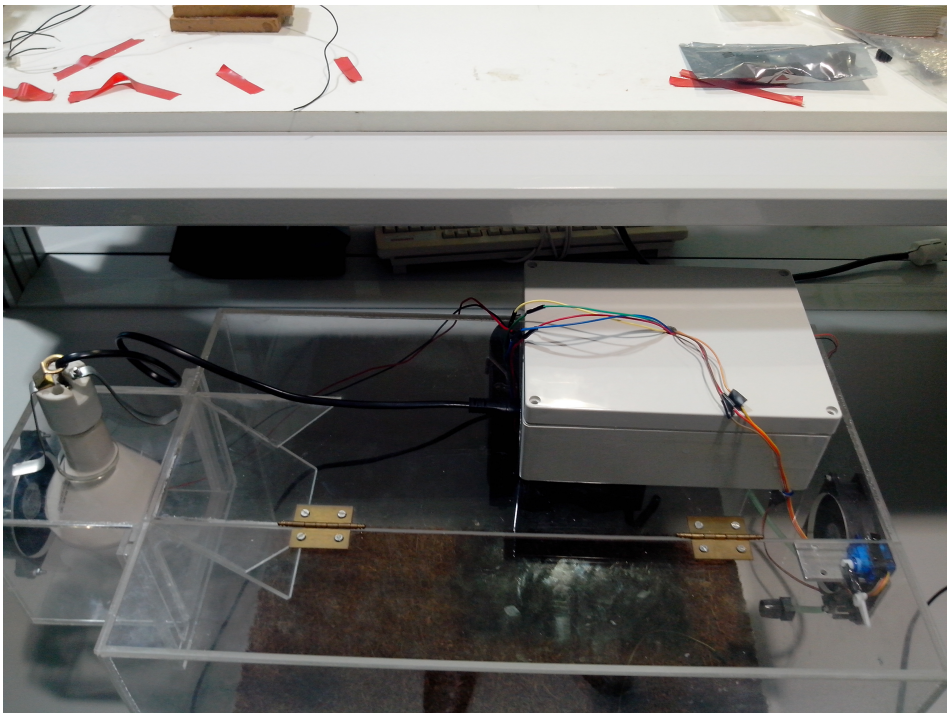


Figura B.1: Agujero para cables de sensor y servo.

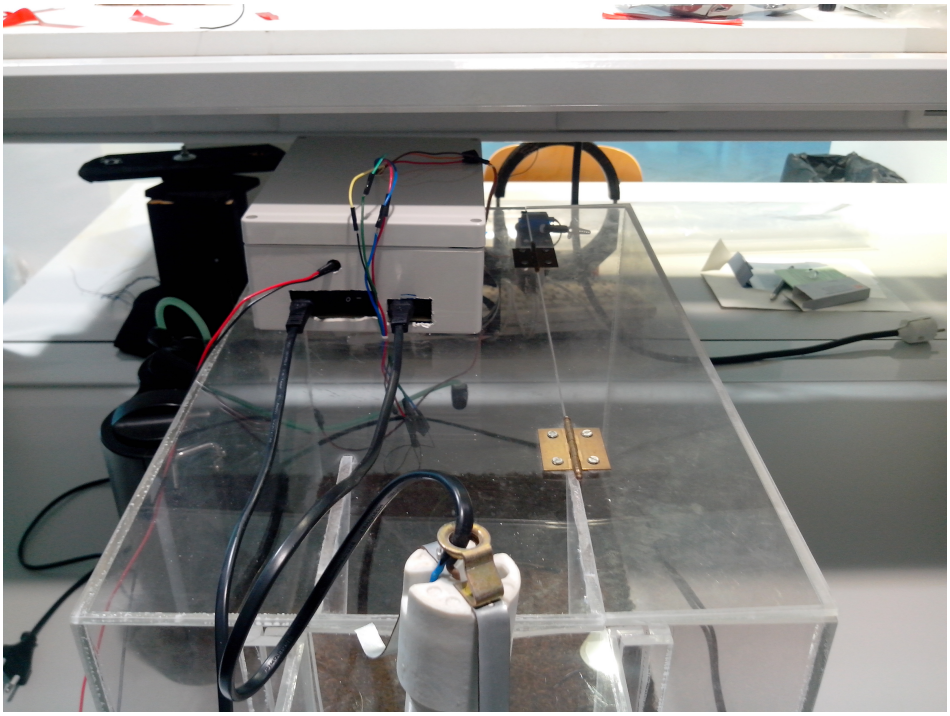


Figura B.2: Agujeros conector AC y bombilla.

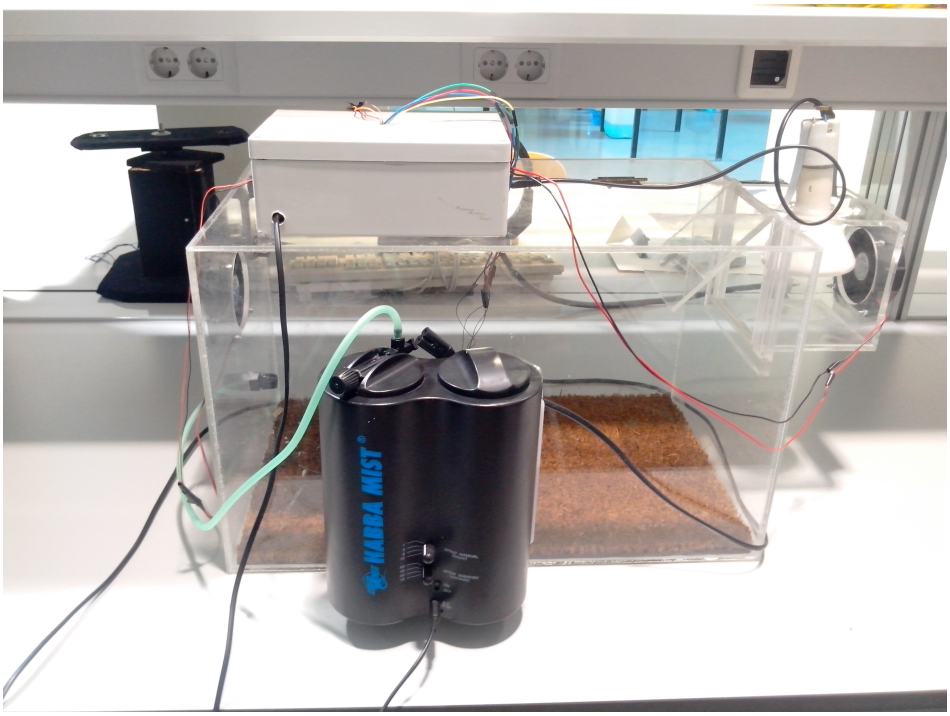


Figura B.3: Conector humedecedor.

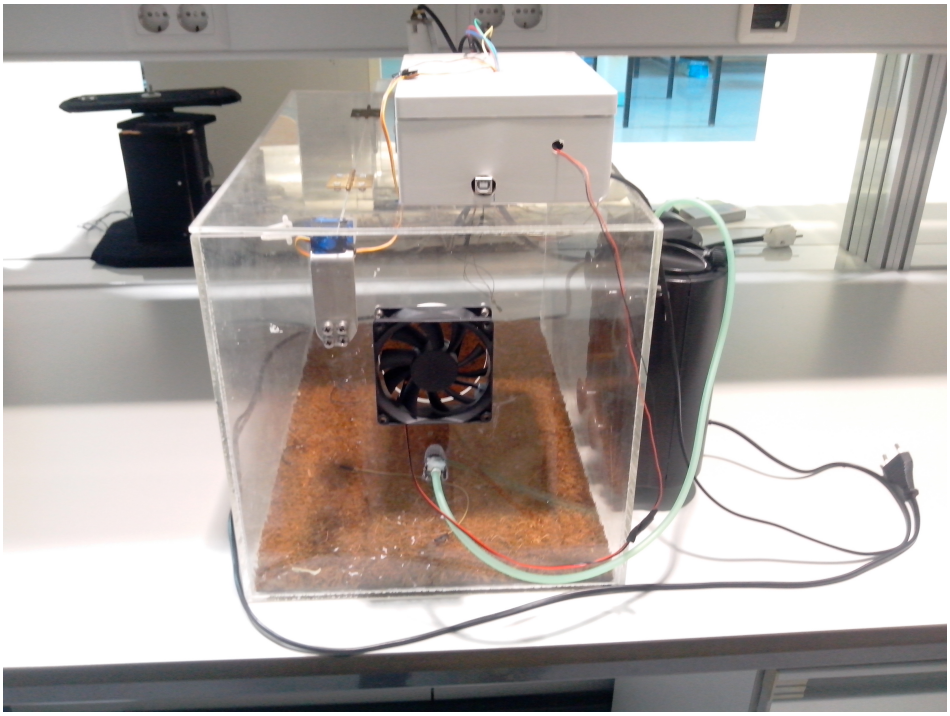


Figura B.4: Conector Arduino